

NASA Technical Memorandum 89119

**ALGORITHM IMPLEMENTATION ON THE
NAVIER-STOKES COMPUTER**

**STEVEN E. KRIST AND
THOMAS A. ZANG**

(NASA-TM-89119) ALGORITHM IMPLEMENTATION ON
THE NAVIER-STOKES COMPUTER (NASA) 76 p
CSCL 01A

N87-20968

G3/02 Unclass
43594

MARCH 1987



National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

Introduction

With the emergence of the general-purpose supercomputer, it has become possible to perform accurate numerical simulations of a wide variety of complex, three-dimensional scientific problems. Yet there are many problems of interest which are so complex that, in order to simulate them numerically, supercomputers with computation rates and storage capacities orders of magnitude larger than those of current supercomputers are needed. Furthermore, the computational requirements of such problems will continue to surpass the performance of general-purpose supercomputers for at least the next decade. As a result, increased attention has been directed towards an emerging class of parallel-processing supercomputers.

One such parallel-processing supercomputer is the "Navier-Stokes Computer" (NSC) [1, 2], which is being developed by Daniel Nosenchuck and Michael Littman at Princeton University. The NSC consists of multiple (up to 128) local memory parallel processors, called Nodes, interconnected in a hypercube network. Each Node has the power and speed of a Class VI supercomputer, giving a projected, full 128 Node NSC a storage capacity in excess of 262 Gbytes, and a peak speed in excess of 61 GFLOPS.

The computational speed and efficiency of the NSC are derived in part from a Nodal architecture which provides parallel operation of both the memory and the computational units. Parallel operation of the memory is attained by distributing the local memory of a Node over multiple interleaved memory planes, all of which may operate simultaneously. Multiple operand vectors may then be streamed from memory into a Reconfigurable Arithmetic and Logic Unit

(RALU), while multiple result vectors are simultaneously routed from the RALU back to memory. The RALU architecture involves the interconnection of multiple high-speed floating point and logic processing units into single or multiple parallel pipelines. Since the RALU is dynamically reconfigurable, many different parallel pipeline configurations may be formed from the processing units. Efficient implementation of an algorithm on the NSC thus entails the effective utilization of not only the coarse grain parallelism associated with distributing the problem over the Nodes, but also of the fine grain parallelism associated with configuring the parallel pipelines and with routing multiple vectors between the RALU and memory.

Although the NSC was originally designed for the numerical simulation of flows governed by the Navier-Stokes equations, it provides a means for efficiently solving most any problem for which the computation is numerically intensive, and for which the algorithm makes use of long vectors and may be parallelized with a coarse granularity using local rather than shared memory. The objective of this paper is to detail the procedures involved in implementing one such algorithm on the NSC. Particular attention is focused on procedures for mapping the computational grid into the Nodes, for configuring the RALU pipelines, and for allocating memory planes for storing variables such that multiple vectors are efficiently routed between memory and the RALU. The resulting analysis constitutes virtually an assembly language level description of algorithm implementation on the NSC.

The specific finite-difference algorithm considered herein was developed for the direct numerical simulation of laminar-turbulent transition in wall-bounded shear flows by solution of the incompressible Navier-Stokes equations.

Recent work in this area [3, 4, 5] indicates that numerical algorithms for performing this simulation are quite mature. However, it has not been possible to continue the simulations up through the onset of fully turbulent flow, since current supercomputers do not have the storage capacity and speed needed to satisfy the extreme resolution demands of the later transition stages. Furthermore, current transition simulations must resort to using the parallel flow assumption, because the resolution demands of a true spatially developing flow are an order of magnitude more extreme. Hence, the numerical simulation of laminar-turbulent transition exemplifies the need for new supercomputers like the NSC, which have the power and speed requisite for performing these simulations.

In formulating the algorithm, the Navier-Stokes equations are temporally discretized using a splitting technique in conjunction with low storage Runge-Kutta treatment of the advection term, Crank-Nicholson treatment of the diffusion term, and backward Euler treatment of the pressure term. The resulting algorithm requires the solution of a Helmholtz equation for each velocity component and of a Poisson equation for the pressure, at each Runge-Kutta stage. Solution of these equations, which constitutes the bulk of the computational work, is analyzed for two iterative methods, namely, Red-Black SOR and ZEBRA 1. Red-Black SOR is a two-color explicit point method in which a point Jacobi type iteration is applied at alternating points. ZEBRA 1 is a two-color implicit line method in which line relaxation is applied along alternating lines.

It should be emphasized that the purpose of this analysis is to detail the procedures involved in implementing a representative algorithm for simu-

lating the laminar-turbulent transition problem on the NSC. The algorithm considered in this analysis was chosen for its simplicity, particularly in regard to the use of second-order discretization methods and single grid iterative methods. More suitable algorithms will surely use higher order discretization methods, and employ conjugate gradient or multigrid methods based on the single grid iterative methods considered here. However, the implementation procedures detailed herein are indicative of the procedures required for implementing the more complex algorithms on the NSC.

In the two following sections, architectural details of the NSC and formulation of the algorithm are briefly discussed. Details of the procedures for implementing the algorithm on the NSC, and projected timing results for a particular problem, are presented in the subsequent section. Some concluding remarks are presented in the final section.

Architectural Overview of the NSC

The NSC is a multi-purpose parallel-processing supercomputer which is designed to perform numerical simulations of a wide variety of large, numerically intensive, complex scientific problems. Rapid solution of these problems is attained through a global architecture which distributes the computations over a fairly small number of powerful local memory parallel processors, called Nodes. A broad overview of the global and Nodal architectures is presented below.

As the NSC is currently in the developmental stage, certain architectural details have not been finalized. For instance, tradeoffs between having the Nodal memory distributed over 8 memory planes rather than 16 memory planes (as assumed herein), are still under review. However, such alterations in the architectural details are likely to affect the algorithm implementation procedures to only a limited extent. It should also be noted that the modular design of the NSC allows for relatively easy upgrading of the hardware components. Hence, the memory/speed characteristics of the final design may differ from the characteristics detailed herein, which are based on the utilization of mid-1986 technology.

Global architecture. - The global architecture of the NSC involves the interconnection of the Nodes through two communication networks. The first network utilizes a global drop-line bus to link the entire Node array to a front-end computer. Although the global bus is primarily used to transfer data and commands between the front-end and the Nodes, it may also be used to transfer data between any two Nodes of the array. The second communication network involves the interconnection of the Nodes in a hypercube network [6]. Internode communication links for the hypercube network are implemented with fiber-optic transmission lines, providing data transmission rates orders of magnitude faster than those provided by the global bus. Consequently, during execution of the calculation procedures for a given algorithm, most (if not all) internode data transfers are routed through the hypercube network. A schematic of the global architecture, where a subset of the Nodes and a simple 2-D nearest neighbor interconnect network are illustrated, is presented in figure 1.

The front-end is a general-purpose computer which provides the operating environment for the NSC. In-line it is used to load data and commands to the Nodes and to monitor the Node array. Off-line it provides program development, work station support, and data analysis capabilities.

Nodal architecture. - The Nodal architecture is designed to permit parallel operation of both the memory and the computational units, providing large throughput rates for a wide variety of computational procedures. Architecturally central to this operation are the multiplane interleaved memory, the dynamically reconfigurable ALU (RALU), and the Memory-ALU-Switch Network (MASNET). The interconnection of these devices is illustrated in figure 2. Computations are begun by streaming multiple operand vectors from the memory planes to the input ports of MASNET. The operand vectors are then routed through MASNET, which essentially is a 16x16 nonblocking switch, and enter the input ports of the RALU. Result vectors from the RALU are then routed back through MASNET and sent to the appropriate memory planes for storage.

The multiplane interleaved memory consists of sixteen 128 Mbyte memory planes, giving each Node a local memory of 512 Mwords for 32-bit words. During a given process, each memory plane may be connected to either an input port or an output port of MASNET, but not both. Consequently, a memory plane cannot be used to store elements of a result vector while simultaneously being accessed for elements of an operand vector. Elements of the vectors are stored in and accessed from the memory planes using either constant stride or scatter-gather addressing. Each of the memory planes has complete address translation capabilities to provide full "scatter-gather" capability. The address translation information is stored in high-speed look-up tables.

The RALU is a pipeline which performs multiple sequential and/or parallel operations on vectors. It contains twenty-four floating-point processing elements, eight logic units, and various pipeline support hardware elements which are interconnected to form the processing pipelines for particular computational procedures. The entire pipeline is reconfigurable within one clock period, which provides a dynamically changing vector processing environment.

Of the floating-point processing elements, twenty-two elements perform addition, subtraction, multiplication, floating to fixed-point, or fixed to floating-point operations. In addition, upon performing one of the above operations, the element may also be used to change the sign of the result, or take the absolute value of it. For these elements, the startup cost associated with processing the first value in a vector is one clock period. Each of the other two elements actually consists of four chips which are pipelined to form a single element. These elements may be used to perform division, evaluate transcendental functions, or take the square root of the input values. The startup cost for these elements is four clock periods. The nominal operation rate of the processing elements is 20 MFLOPS, providing a Nodal peak speed of 480 MFLOPS.

Associated with each of these elements is a constant parameter latch. These latches are used to input constants to the elements, and may be set with either ALU or scalar results. It takes two clock periods to reset a constant parameter latch, and only one latch may be reset at a time.

The logic units may be used for fixed point addition, subtraction, multiplication, or division, or to perform logical comparisons on either fixed or

floating-point values. The startup cost for the logic units is four clock periods for performing division, or one clock period for performing any of the other operations. As with the floating-point processing elements, the operation performed by a logic unit may be changed every clock period.

An illustration of how the hardware elements in the RALU are configured to form the parallel processing pipelines is provided in figure 3. This particular pipeline performs a point Jacobi iteration of the central-differenced 3-D Poisson equation,

$$\nabla^2 P = G ,$$

on a uniform grid. Here, fifteen of the available floating-point processing elements and one logic unit have been configured in a tree structure. Nine operand vectors are accepted in parallel at the pipeline entrance. Operations on these vectors are then performed in parallel, with results from the operations immediately routed to subsequent levels of the pipeline. Note that once the residual value, R_{ijk} , is available, it is routed simultaneously to the remainder of the pipeline for updating P , and to a pipeline for performing a local convergence check. In this example, the convergence criteria is $|R_{ijk}| < \epsilon$. Upon satisfaction of this condition at all grid points, the convergence flag interrupts the microsequencer, and the pipeline is reconfigured for execution of the next calculation procedure in the algorithm. The nominal operation rate for execution of this point Jacobi iteration procedure is 300 MFLOPS.

As alluded to above, the processing elements and logic units, along with various pipeline support hardware elements, are configured into specific pipelines by a microsequencer. In general, this unit is used to configure the

pipeline at the start of an array operation, and the pipeline remains fixed until completion of that operation. However, results from logical comparison operations may be used to conditionally reconfigure the pipeline at every clock period, without (in general) requiring a pipeline flush. This capability permits the vectorization of many powerful algorithms which are not vectorizable on conventional vector architecture supercomputers.

It should be noted that since the NSC is still under development, specific details of the RALU architecture, and thus the degree to which the RALU may be reconfigured, have not been finalized. Indeed, a major purpose of this study was to identify specific pipelines to be included in the final design. Therefore, it is assumed in subsequent sections of this paper that the only constraint in forming a pipeline is that the number of elements utilized may not exceed the number of elements to be contained in the RALU.

As mentioned previously, the flow of vectors between memory and the RALU is controlled by the Memory-ALU-Switch Network (MASNET). MASNET consists of fifty-six 32-bit switch elements, where each element has two input ports, two output ports, and a bidirectional port to a 32-word local memory. Words may be stored in the local memory of an element and delayed for up to 16 clock periods when two vector streams are input, or up to 32 clock periods when a single vector stream is input. MASNET is formed by the interconnection of the switch elements into a 16x16 nonblocking switch. The entire switch may be rerouted every clock period. MASNET is used to route operands from memory to the RALU inputs, route results from the RALU to memory inputs and/or RALU pipeline inputs (for vector recursion), and to transfer memory contents between memory planes. MASNET is also utilized for internode communications, which are implemented by transferring data between each Node's MASNET.

Another important feature of MASNET is that it permits the generation of multiple, parallel vector streams from a single source. In this process, as values which constitute a given vector are routed through a switch element, they are simultaneously stored in the local memory of the element. During subsequent clock periods, the values are retrieved from memory and routed out of the switch through the second output port, as a second vector. Since the order in which the values are retrieved need not be the same as the order in which they are stored, the generated vector may deviate in local ordering from that of the original vector. By applying this process over a series of switch elements, multiple vectors may be generated from a single source.

A second means for generating multiple vectors from a single source involves utilization of the vector delay and regeneration unit. This unit is located between the MASNET outputs and RALU inputs. It may be used to perform the same process described above, except that the ordering of values in the generated vectors must be the same as that of the original vector. However, whereas MASNET may be used to delay vectors for 224 clock periods at most, the vector delay and regeneration unit may be used to delay vectors for thousands of clock periods.

The operation of the Node is controlled and monitored by a Node manager. The Node manager is primarily used as an intelligent interface between the Node and the front-end. It provides initialization, checkpointing, and data store handling capabilities, and decodes "macromachine instructions" which are used to configure the RALU. In addition, the Node manager provides scalar operation capabilities at a rate of 2 MFLOPS. However, it should be emphasized that the Node manager rarely becomes involved in numerical computations other than for evaluating expressions for use as constants in the RALU.

Internode communication. - Internode addressing on the NSC is supported by two addressing modes, global addressing and explicit boundary point definition (BPD). In this analysis, only BPD addressing is considered. BPD involves the explicit definition of all boundary-point data (i.e., data associated with points on the boundary of the computational subdomain, which are to be transferred to other Nodes): the source Node of the data, and all destination addresses.

The internode communication process begins as results enter the first row of switch elements in MASNET. If a result has been defined as a boundary-point value, it is routed to the boundary cache of the source Node while simultaneously being routed through the switch element. The boundary cache is linked to all of the switch elements in the first row of MASNET by a common bus. The clock for the common bus (and the "hyperspace router") runs four times faster than the clock for MASNET and the RALU. Thus, every clock period, it is possible to route a boundary value from each of four switch elements to the boundary cache. Once the data is received in the boundary cache, it is immediately routed to the local "hyperspace router" of the Node, and sent to the boundary cache of the destination Node. Then when boundary-point data is needed in an ongoing computation of the destination Node, it is accessed from the boundary cache of the destination Node and inserted into MASNET at the last row of switch elements. As with the first row of switch elements, all the elements in the last row of MASNET are linked to the boundary cache by a common bus. A schematic of the hardware interconnects between MASNET, the boundary cache, and the hyperspace router is presented in figure 4.

The local hyperspace routers are nonblocking permutation switch networks which are used to route boundary-point data to the appropriate internode communication links. The data are self routing in that the destination addresses, which are carried with the data, are used to set the hyperspace router switch states. For a 128 Node NSC, the hyperspace routers contain 8x8 switch networks, and the hypercube internode communication network links each Node to seven neighboring Nodes. Although internode communication is most easily accomplished for data transfers in which the destination Node is directly linked to the source Node, data may be transferred between any two Nodes by routing the data over a series of hyperspace routers.

The internode communication links are implemented with fiber-optic cables, providing data transmission in byte-serial format at a duplex rate of 1 Gbyte/sec. The boundary cache of each Node consists of a 1 M-word write-through cache. For BPD addressing, the boundary cache is continuously updated by pre-communicating the boundary point data as it is generated in the source Node. Thus, current boundary data is usually maintained within each Node, which eliminates most of the internode communication overhead.

Internode communication delays, which result in the temporary suspension of computations, may occur in a number of situations. One such situation is when boundary-point values are not present in the boundary cache of the Node at the time they are required in the ongoing computation. The computations must then be suspended while those values are retrieved from other Nodes. This type of delay is likely to occur in problems for which the amount of BPD data required by each Node is greater than the storage capacity of the boundary cache. A second situation for which communication delays occur is when

the operands for computing a particular term include more than four boundary-point values. In this case, since only four values may be accessed from the boundary cache each clock period, routing of the operands must be delayed for at least one clock period while all of the boundary-point values are inserted into MASNET.

Delays may also occur in routing the BPD data out of the hyperspace routers. This situation arises during burst transmissions where a significant portion of the BPD data is transferred between Nodes which are not directly linked by the hypercube network. Since the BPD data must then be routed over a series of hyperspace routers, the switch networks of the routers may become overloaded. If the overloading is severe enough, the BPD data will not be present in the boundary cache of the destination Node at the time it is required in an ongoing computation, causing a temporary suspension in the computations. This type of internode communication delay is most likely to occur in algorithms where the computations are not localized (e.g. global spectral methods). For algorithms in which the computations are localized (e.g. finite-difference methods), the amount of data which must be transferred between Nodes is small enough that delays in routing the data out of the hyperspace routers are unlikely to cause a suspension in the computations, particularly since the data is pre-communicated.

Numerical Simulation of Laminar-Turbulent Transition

The particular problem considered in this analysis is the numerical simulation of laminar-turbulent transition in plane Poiseuille flow. The channel flow geometry is illustrated in figure 5. The governing equations are the incompressible time-dependent Navier-Stokes equations with constant viscosity. Employing the usual scaling for the channel, the lengths are scaled by the channel half width, l , and the velocities are scaled by the centerline velocity for the mean flow U_0 . Written in rotation form, the nondimensionalized equations are

$$\begin{aligned} \underline{u}_t &= \underline{u} \times \underline{\omega} - \nabla P + (1/Re) \nabla^2 \underline{u} - f \hat{i} \\ \nabla \cdot \underline{u} &= 0 \end{aligned} \tag{1}$$

where

$$\begin{aligned} \underline{\omega} &= \nabla \times \underline{u} \\ P &= p + 1/2 |\underline{u}|^2 \\ f &= (1/Re) \frac{\partial^2 u_0(y)}{\partial y^2} \end{aligned}$$

$u_0(y)$ denotes the mean flow velocity, and $Re = U_0 l / \nu$ the Reynolds number. The forcing function f represents the mean pressure gradient which drives the mean flow.

Boundary conditions implied in this formulation are the no-slip condition at the walls, and periodic boundary conditions in the spanwise direction. For a true spatially developing flow, appropriate inflow, and outflow boundary conditions are applied in the streamwise direction. However, determination of an appropriate outflow boundary condition (a problem which has yet to be

The temporal discretization of the above equations involves a low storage second-order Runge-Kutta treatment of the advection term and a Crank-Nicholson treatment of the diffusion term in the velocity step, with a backward Euler pressure correction applied after each Runge-Kutta stage. For simplicity, second-order Runge-Kutta is used in this analysis. However, higher order Runge-Kutta schemes may be incorporated with only minor modifications to the solution procedure detailed herein. With minor changes in notation from eqs. (2) and (3), the time discretized equations may be written as

$$\underline{u}^* - (h/4\text{Re}) \nabla^2 \underline{u}^* = \underline{u}^n + (h/2) \underline{F}^n + (h/4\text{Re}) \nabla^2 \underline{u}^n \quad (4)$$

$$\begin{aligned} (2/h) (\underline{u}^{n+1/2} - \underline{u}^*) &= -\nabla p^{n+1/2} \\ \nabla \circ \underline{u}^{n+1/2} &= 0 \end{aligned} \quad (5)$$

$$\underline{u}^{**} - (h/4\text{Re}) \nabla^2 \underline{u}^{**} = \underline{u}^{n+1/2} + h(\underline{F}^{n+1/2} - 1/2 \underline{F}^n) + (h/4\text{Re}) \nabla^2 \underline{u}^{n+1/2} \quad (6)$$

$$\begin{aligned} (2/h) (\underline{u}^{n+1} - \underline{u}^{**}) &= -\nabla p^{n+1} \\ \nabla \circ \underline{u}^{n+1} &= 0 \end{aligned} \quad (7)$$

where $\underline{F} = \underline{u} \times \underline{\omega} - \hat{f}\hat{i}$, and h denotes the time increment. In order to make the pressure correction step amenable to solution, the divergence of the pressure equation is taken and the incompressibility constraint is enforced on it. Absorbing the time increment into the pressure value, then from eq. (5),

$$\nabla^2 p^{n+1/2} = \nabla \circ \underline{u}^* \quad (8)$$

(Equation 8 is the consistent Poisson equation. It represents the composition of the discrete divergence operating on the discrete gradient of the

resolved adequately) is particularly difficult in that conditions at the boundary are not known a priori. Although the outflow boundary conditions utilized in previous low resolution simulations of spatially developing shear flows have a relatively small upstream influence on the solution [7, 8], it is unclear as to whether these boundary conditions are adequate for a full laminar-turbulent transition simulation. If the parallel flow assumption is made, periodic boundary conditions are applied in the streamwise direction. The initial condition is that of a small disturbance superimposed upon fully developed plane Poiseuille flow.

Algorithm formulation. - The solution algorithm is based on a time-splitting scheme in which the solution is advanced from $t = t^n$ to $t = t^{n+1}$ as follows: in the first step,

$$\underline{u}_t = \underline{u} \times \omega + (1/Re) \nabla^2 \underline{u} - f\hat{i} \quad (2)$$

is integrated from t^n to the intermediate time t^* ; in the second step,

$$\begin{aligned} \underline{u}_t &= -\nabla P \\ \nabla \circ \underline{u} &= 0 \end{aligned} \quad (3)$$

is integrated from t^* to t^{n+1} . Application of the boundary conditions for this scheme is discussed in detail by Zang and Hussaini [9], and will not be covered here. However, it should be noted that whereas a staggered grid is used for the pressure in the formulation of [9], a non-staggered grid is used in this formulation. Use of the non-staggered grid not only changes the form of the spatially discretized equations, but also necessitates the application of a consistent artificial boundary condition for the pressure [10].

pressure.) Upon solution of this Poisson equation for the pressure, the velocity is corrected from

$$\underline{u}^{n+1/2} = \underline{u}^* - \nabla p^{n+1/2} \quad (9)$$

The pressure correction step after the second Runge-Kutta stage is treated in a similar manner.

Formulation of the algorithm is completed by spatially discretizing the equations. Since the NSC is tailored towards algorithms in which the computations are localized, central-differencing is used in the spatial discretization of this formulation. The computational grid for this problem is Cartesian with uniform spacing in the streamwise and spanwise directions, and arbitrary stretching in the vertical direction. Grid stretching is used in the vertical direction so that grid points can be concentrated near the walls, where gradients in the primitive variables are especially large.

Solution procedure. - The procedure for advancing the solution from $t = t^{n+1}$ begins with the calculation of the three components of the advection term,

$$\underline{F}^n = \underline{u}^n \times \underline{\omega}^n - f\hat{i} \quad (10)$$

The second step of the procedure entails calculating the right hand side of the Helmholtz equation for the x velocity component, and then solving the Helmholtz equation to update u to the intermediate time level t^* . This procedure is then repeated for the y and z velocity components. Denoting the right hand sides of the three Helmholtz equations as \underline{L} , then the equations are

$$\underline{L}^n = \underline{u}^n + (h/2) \underline{F}^n + (h/4Re) \nabla^2 \underline{u}^n \quad (11)$$

$$\underline{u}^* - (h/4Re) \nabla^2 \underline{u}^* = \underline{L}^n \quad (12)$$

Upon updating the velocity components, the right hand side of the Poisson equation is calculated, and then the Poisson equation is solved for $P^{n+1/2}$, where

$$K^* = \nabla \circ \underline{u}^* \quad (13)$$

$$\nabla^2 P^{n+1/2} = K^* \quad (14)$$

Corrected values of the velocity components are then determined from

$$\underline{u}^{n+1/2} = \underline{u}^* - \nabla P^{n+1/2} \quad (15)$$

The above procedure is then repeated for the second stage of the algorithm to advance the solution to t^{n+1} . The only difference in the second stage calculations is that F^n is absorbed in the calculation of $\underline{F}^{n+1/2}$, as

$$\underline{F}^{n+1/2} = \underline{u}^{n+1/2} \times \underline{\omega}^{n+1/2} - \hat{f}i - (1/2) \underline{F}^n \quad (16)$$

As mentioned previously, two iterative methods for solving the Helmholtz and Poisson equations are considered in this analysis. The first method is Red-Black SOR. This is an explicit two-color point method in which a point Jacobi type iterative scheme is utilized. A complete iteration involves updating red points ($i+j+k$ odd) first, and then updating black points ($i+j+k$ even) using the latest available values in calculation of the residual.

The second method is ZEBRA 1, which was developed as a means for performing line relaxation in a vectorizable manner [11]. It is a two-color line

scheme in which line SOR is performed along alternating lines, where the grid in planes normal to those lines has a checkerboard pattern. In this algorithm the line relaxation is performed in the vertical direction, which is the direction of grid stretching. The tridiagonal systems of equations generated for the vertical lines are first solved along red lines ($i+j$ even), and then solved along black lines ($i+j$ odd) using the latest available values in the residual calculation. The tridiagonal systems of equations are solved using the Thomas algorithm.

This section of the paper is concluded with a brief discussion on enforcement of the boundary conditions, since this subject has been avoided up to now. In many algorithms, and in this algorithm in particular, enforcement of the boundary conditions involves minor modifications to the discrete equations at the boundaries. These modifications generally consist of either adding or deleting a few terms from the discrete equations, without changing the form of the equations. On the NSC, incorporation of such modifications to the RALU pipelines involves a reconfiguration in which a few processing elements are either added to, or deleted from, the pipeline. Since the RALU is reconfigurable every clock period, in most cases the reconfiguration may be performed without interrupting the routing of operands through the pipeline. Therefore, on the NSC, the treatment of boundary conditions has little or no effect on the implementation of computational procedures, or on the computation rate. For this reason, and in order to keep the remainder of this analysis as simple as possible, the treatment of boundary conditions is avoided in subsequent sections.

Implementation of the Algorithm

Implementation of the aforementioned algorithm begins with a determination of how the computational domain is to be distributed over the Nodes. For now it is assumed that an $L \times M \times N$ computational grid is subdivided into 128 subdomains of dimension $I \times J \times K$, where $I = L/8$, $J = M/4$, and $K = N/4$. In order to simplify the computational procedures, it is further assumed that I is an even number, and that J and K are multiples of 4. Conceptually, the computational domain is mapped into a three-dimensional lattice of Nodes. For finite-difference algorithms then, each interior Node need communicate only with its adjacent Nodes. Boundary Nodes (i.e. Nodes into which grid points from the boundary of the computational domain have been mapped) must also communicate with non-adjacent Nodes in situations where periodic boundary conditions are to be enforced. The hypercube network does not provide direct links between all adjacent Nodes in a three-dimensional lattice, or between apposite non-adjacent boundary Nodes.

With this mapping of the computational domain, the Nodes typically perform identical processes. In fact, the only situation for which processes differ is when boundary Nodes are evaluating terms in which the boundary conditions are enforced. As discussed in the section on algorithm formulation, treatment of the boundary conditions requires only minor modifications to the RALU pipelines. Hence, the procedures for implementing the algorithm in the boundary Nodes are nearly identical to the procedure implemented in the interior Nodes.

On the nodal level, a procedure for allocating memory planes to store the variables must be chosen. This allocation procedure is crucial to efficient implementation of the algorithm as it not only affects the ordering of values in the operand and result vectors, but also influences the actual configuration of the RALU pipelines. In the following analysis, the same allocation procedure is used for the Red-Black SOR and ZEBRA 1 solutions of the Helmholtz and Poisson equations. Hence, the procedures for computing the explicit terms in the algorithm are identical for the two iterative methods.

Schematics illustrating the memory plane allocation procedure for storing the primitive variables u , v , w , and p are presented in figures 6, 7, 8, and 9, respectively. Here, two-dimensional subsets of the grid in x - y planes are illustrated. The two numbers above each grid point denote the (i, j) indices of the point; the number below each grid point denotes the memory plane in which the particular variable is stored. Grid points on the dashed lines indicate points which have been mapped into adjacent Nodes. The sequences for allocating the memory planes in the vertical direction are repeated for every fourth x - y plane.

To illustrate the ordering in which a variable is stored within a given memory plane, storage of the primitive variable u within memory plane 0 (MP0) is considered. From figure 6, with k set equal to 1, grid point $(1, 1, 1)$ is the first point for which u is stored in MP0. Consecutive grid points are determined by moving across the grid in x , then up the grid in y . Hence, consecutive memory locations in MP0 contain the values of u for grid points

$(1, 1, 1), (2, 1, 1), (3, 1, 1), \dots, (I, 1, 1),$
 $(1, 5, 1), (2, 5, 1), (3, 5, 1), \dots, (I, 5, 1),$
 $(1, J-3, 1), (2, J-3, 1), (3, J-3, 1), \dots, (I, J-3, 1)$

Moving up the grid to plane $k=3$, the next set of consecutive memory locations in MP0 contain the values of u for grid points

$(1, 3, 3), (2, 3, 3), (3, 3, 3), \dots, (I, 3, 3),$
 $(1, 7, 3), (2, 7, 3), (3, 7, 3), \dots, (I, 7, 3),$
 $(1, J-1, 3), (2, J-1, 3), (3, J-1, 3), \dots, (I, J-1, 3)$

This sequence for storing the values of u is repeated for planes $k=5$ and 7 , $k=9$ and 11 , etc., up through planes $k=K-3$ and $k=K-1$. It follows that the values of u for one-eighth of the grid points are stored in MP0. The remaining values of u are distributed over MPs 1-7, using similar procedures to assign the array elements to consecutive memory locations.

Comparison of figures 6, 7, 8, and 9 indicates that at every grid point for which u is stored in MP0, the values of v , w , and P are stored in MPs 8, 2, and 10, respectively. Consequently, the sequence for storing the values of v, w , and P at these grid points is identical to the sequence for storing u . Similar relationships between the variables, and the memory planes in which the variables are stored, hold at all other grid points.

The manner in which the variables are stored suggests a natural ordering for values in the operand and result vectors. In general, the sequential order of values in a given vector is the same as the order in which those

values are stored. This natural ordering gives vector lengths of $I \times J \times K / 8$. There are two occasions when orderings other than the natural ordering occur. One occasion is when elements of an operand vector are accessed as BPD data from neighboring Nodes, in which case those values are inserted into the vector as the operands stream through MASNET. The other occurrence is in the iterative solution of the Helmholtz and Poisson equations, as will be discussed later in this section.

Calculation of the Advection Term. - The first term to be calculated is the advection term for the first stage of the Runge-Kutta/Crank-Nicholson temporal discretization. Using second-order central differencing in the spatial discretization of Eq. (10), and denoting the (x, y, z) components of the advection term and the vorticity vector as $\underline{F} = (F, G, H)$ and $\underline{\omega} = (\xi, \eta, \zeta)$, the components of the advection term may be written as

$$F_{ijk}^n = v_{ijk}^n \zeta_{ijk}^n - w_{ijk}^n \eta_{ijk}^n - f \quad (17)$$

$$G_{ijk}^n = w_{ijk}^n \xi_{ijk}^n - u_{ijk}^n \zeta_{ijk}^n \quad (18)$$

$$H_{ijk}^n = u_{ijk}^n \eta_{ijk}^n - v_{ijk}^n \xi_{ijk}^n \quad (19)$$

where

$$\xi_{ijk}^n = (w_{ij+1k}^n - w_{ij-1k}^n) / 2\Delta y - (dh_k v_{ijk+1}^n + dm_k v_{ijk}^n + dl_k v_{ijk-1}^n) \quad (20)$$

$$\eta_{ijk}^n = dh_k u_{ijk+1}^n + dm_k u_{ijk}^n + dl_k u_{ijk-1}^n - (w_{i+1jk}^n - w_{i-1jk}^n) / 2\Delta x \quad (21)$$

$$\zeta_{ijk}^n = (v_{i+1jk}^n - v_{i-1jk}^n) / 2\Delta x - (u_{ij+1k}^n - u_{ij-1k}^n) / 2\Delta y \quad (22)$$

and dh_k , dm_k , and dl_k denote the coefficients for the first derivative in the stretched coordinate direction at plane k .

Calculation of the advection term components is performed using a two-step procedure. In the first step, η , ζ , and F are computed at one-eighth of the grid points. The computation begins for the grid point (1, 1, 1). Consecutive values in the result vectors will be for grid points (1, 1, 1), (2, 1, 1), (3, 1, 1), etc. (i.e., the same series of grid points for which u is stored in MP0). From Eqs. (17), (21), and (22), calculation of η , ζ , and F at grid point (i, j, k) requires the values of u_{ijk} , u_{ij+1k} , u_{ij-1k} , u_{ijk+1} , u_{ijk-1} , v_{ijk} , v_{i+1jk} , v_{i-1jk} , w_{ijk} , w_{i+1jk} , and w_{i-1jk} . Looking at an interior grid point associated with the result vectors, say point (2, 5, k) (see figures 6, 7, and 8), the above operands reside in MPs 0, 1, 3, 4, 6, 8, 8, 8, 2, 2, and 2, respectively. Likewise, at grid point (3, 5, k), the operands reside in the next consecutive memory location of their respective memory planes. This indicates that except for those operand values which are accessed as BPD data from adjacent Nodes, the values for a particular operand vector are accessed from consecutive memory locations of the same memory plane.

It should also be noted that the vectors for v_{ijk} , v_{i+1jk} , and v_{i-1jk} are all generated from MP8. The process for generating these vectors, called "vector latching," utilizes the vector delay and regeneration capabilities of MASNET. In the vector latching process, the values streaming out of MP8 constitute the values of the vector for v_{i+1jk} . As these values are routed through one of the switch elements in MASNET, they are also stored in the local memory of the element. One clock period later, the values are routed out of the element as a second vector. This second vector contains the values of v_{ijk} , which have been offset from the v_{i+1jk} values by one value. At the

next switch element, this process is repeated to generate the vector for v_{i-1jk} . Vector latching is also used to generate the vectors for w_{i-1jk} , w_{ijk} , and w_{i+1jk} from MP2.

An illustration of the RALU pipeline configuration for calculating η , ζ , and F is presented in figure 10. Here, the operand memory planes, and the variable stored in each plane, are indicated at the top of the figure. The vector latches for v and w are indicated by the paths the operands take in MASNET. In this 17 operation RALU pipeline, the values of η and ζ are calculated in independent pipelines. Upon calculation of these terms, they are routed both to memory, and to the remainder of the pipeline for calculating F . The results for η , ζ , and F are stored in to MPs 9, 11, and 12 respectively.

The second step of the advection term calculation procedure is to calculate ξ , G , and H at the same grid points for which η , ζ , and F are calculated in the previous step. From Eqs. (18), (19), and (20), calculation of these terms at grid point (i, j, k) requires the values of u_{ijk} , v_{ijk} , v_{ijk+1} , v_{ijk-1} , w_{ijk} , w_{ij+1k} , w_{ij-1k} , η_{ijk} , and ζ_{ijk} . From figures 6, 7, and 8, the values of the first seven operands reside in MPs 0, 8, 12, 14, 2, 3, and 1, respectively. From the previous step, the values of η and ζ reside in MPs 9 and 11, respectively. The 14 operation pipeline for performing the second step computations is illustrated in figure 11. As indicated at the bottom of the figure, the results for G and H are stored in MPs 4 and 13, respectively.

At this point in the computation, the three components of the advection term will have been computed at one-eighth of the grid points. In order to calculate the advection term at the rest of the grid points, this two-step

procedure is then repeated seven times. The sequence in which the first and second step calculation procedures are performed is outlined in tables 1 and 2. Listings of the memory planes in which the operand and result values for each step are stored are also presented in these tables.

In order to project the amount of time it would take for calculation of the advection term, the startup time, actual computation time, and internode communication delay time must be determined. In the various steps of the calculation procedure, once the pipeline is full, and assuming there are no delays, results are generated every clock period. Consequently, for a given step of the procedure the actual computation time, in clock periods, is $I \times J \times K / 8$ (i.e., the length of the vectors). Hence, the actual computation time for the advection term computation is $2 I \times J \times K$ clock periods.

For the first step in the procedure, startup time is accrued as the first values of the operand vectors are accessed from memory and routed through MASNET and the RALU, and as the first values for the result vectors are routed back through MASNET and stored in memory. The initial startup time for this process is

1	to access operands from memory
8	to route operands through MASNET, including the vector latch
7	to route operands through the RALU
7	to route results through MASNET
1	to store results in memory
<u>1</u>	
24	clock periods

Additional delay time is accrued whenever the constant parameter latches which contain the coefficients for the first derivative in the stretched coordinate direction must be reset. For instance, after the operands for grid point (I, J-3, 1) have been processed through the first row of floating-point

processing elements in the RALU, the next operands to be processed in these elements are for grid point (1, 3, 3). Since the value of the k index has changed, the constant parameter latches containing the values of dh_k , dm_k , and dl_k must be reset before the computation can be continued. It takes 6 clock periods to reset these latches each time the computations proceed to another vertical plane. Lumping this delay time with the initial startup time, the total startup time for the first step is $24+6(K/2-1)$ clock periods.

The initial startup time for the second step of the procedure is 22 clock periods. The reason for this delay is that MPs 9 and 11, which are used to store results in step 1, are accessed for operands in step 2. Since a memory plane cannot be accessed for operands while it is being used to store results, operands from MP 9 and 11 cannot be accessed until the last results from step 1 have been stored. Consequently, the computations must be suspended until the first operands for the second step have been routed through MASNET. However, in step 3, none of the memory planes which are accessed for operands have been used to store results in step 2. Therefore, the only initial startup time for step 3 is the time to reset the constant parameter latches. Determining the startup time for the other 13 steps in a similar manner, the total startup time for calculation of the advection term is found to be $162+48K$ clock periods.

In computing the advection term, BPD data for the variables u , v , and w must be present in the boundary caches of the Nodes. Considering the variable u , boundary-point values are required at all $(i, 1, k)$, (i, J, k) , $(i, j, 1)$, and (i, j, K) grid points, for a total of $2(I \times J + I \times K)$ values. Similarly, the number of v and w boundary-point values required are $2(J \times K + J \times I)$ and

$2(KxI+KxJ)$, respectively. Thus, a total of $4(IxJ+JxK+KxI)$ boundary-point values are required for computing the advection term at all of the grid points. Subdomains of dimension $420 \times 420 \times 240$ are utilized in one of the examples presented at the end of this section. For subdomains of this size, computation of the advection term requires around 1.5×10^6 boundary-point values. However, the boundary cache has a 1 Mword storage capacity, so it is not possible to store all of the requisite BPD data within the boundary cache, prior to beginning the computations.

In order to avoid situations where BPD data is not in the boundary cache of the Node at the time it is required, the advection term is computed at one-half of the grid points at a time. Thus, before beginning the computational procedure, the boundary cache of each Node is reloaded with the BPD data for performing the computations at the first half of the grid points (i.e. steps 1-8 in tables 1 and 2). For an interior Node, this requires the communication of $2(IxJ+JxK+KxI)$ boundary-point values to its adjacent Nodes. Utilizing the scatter-gather capabilities of the Nodal memory, and accessing the BPD data from four memory planes at a time, the delay for this process is $1/2(IxJ+JxK+KxI)$ clock periods. Upon completion of the computations at the first half of the grid points, computations are suspended while the boundary caches are reloaded with the BPD data for computations at the second half of the grid points. The total internode communication delay for reloading the boundary caches is $IxJ+JxK+KxI$ clock periods. There are no other internode communication delays in the advection term calculation procedure.

Combining the startup time, actual computation time, and internode communication delay time, the total time for computation of the advection term is

164+(48+2IxJ)K+IxJ+JxK+KxI clock periods. The operation count for the procedure is 31 IxJxK. Neglecting the startup cost and internode communication delay cost, the nominal operation rate of the procedure is 310 MFLOPS.

Calculation of the Remaining Explicit Terms. - The next term to be calculated is the right hand side of the Helmholtz equation for the x velocity component. Using second-order central differencing in the spatial discretization of Eq. (11), and denoting the components of \underline{L} as $\underline{L} = (L, M, N)$, then the x velocity component of the equation may be written as

$$\begin{aligned} L_{ijk}^n = & u_{ijk}^n + (h/2)F_{ijk}^n + (h/4Re) ((u_{i+1jk}^n + u_{i-1jk}^n)/\Delta x^2 \\ & + (u_{ij+1k}^n + u_{ij-1k}^n)/\Delta y^2 + Dh_k u_{ijk+1}^n + B_k u_{ijk}^n + D\ell_k u_{ijk-1}^n) \end{aligned} \quad (23)$$

where

$$B_k = Dm_k - 2/\Delta x^2 - 2/\Delta y^2$$

and Dh_k , Dm_k , and $D\ell_k$ denote the coefficients for the second derivative in the stretched coordinate direction.

The first step of the calculation procedure is to compute the values of L for the vector beginning at grid point (1, 1, 1). From figure 6, the values of u for this computation reside in MPs 0, 1, 3, 4, and 6. As indicated in table 1, the values for F are accessed from MP 12. The 15 operation RALU pipeline for calculating the right hand side of the Helmholtz equation is illustrated in figure 12. As indicated at the bottom of the figure, the results for L are stored in MP 8.

The sequence for the remaining 7 steps of the calculation procedure, and the memory planes utilized in each step, are listed in table 3. The startup time and actual computation time for the procedure are 18+24K clock periods

and $I_x J_x K$ clock periods, respectively. The internode communication delay time, which is accrued while the boundary cache is reloaded with the $2(I_x J_x + J_x K + K_x I)$ boundary-point values of u required in the computations, is $1/2(I_x J_x + J_x K + K_x I)$ clock periods. Note that for subdomains of dimension $420 \times 420 \times 240$, the storage capacity of the boundary cache is large enough to store all of the BPD data for u . The operation count for this procedure is $15 I_x J_x K$, and the nominal operation rate is 300 MFLOPS.

The next step in the solution algorithm is to solve the Helmholtz equation for the x velocity component. The Red-Black SOR and ZEBRA 1 iterative procedures for solving the Helmholtz equation are discussed later in this section. Upon solution of this equation, the next steps in the algorithm are to compute the right hand side of the Helmholtz equation, and then solve the equation, for the y velocity component, and then for the z velocity component. The procedures for computing the right hand side of the Helmholtz equation for the y and z velocity components are identical to the procedure for the x velocity component, although different memory planes as utilized in the various steps of the procedures. The total time to compute the right hand sides of the three Helmholtz equations is $54 + (72 + 3I_x J_x)K + 3/2(I_x J_x + J_x K + K_x I)$ clock periods, and the operation count is $45 I_x J_x K$.

The next step in the algorithm is to compute the right hand side of the Poisson equation for pressure. Spatially discretizing Eq. (13), the term may be written as

$$\begin{aligned}
 K_{ijk}^* = & (u_{i+1jk}^* - u_{i-1jk}^*)/2\Delta x + (v_{ij+1k}^* - v_{ij-1k}^*)/2\Delta y \\
 & + d_h w_{ijk+1}^* + d_m w_{ijk}^* + d_l w_{ijk-1}^*
 \end{aligned} \tag{24}$$

In computing the values of K , 7 operands are required and one result is generated, using 8 of the MASNET output ports. Since 16 output ports are available, it is possible to compute this term for two distinct vectors at the same time. The 22 operation RALU pipeline for performing this computation is illustrated in figure 13. Here, the left side of the pipeline computes the values of K for the vector beginning at grid point (1, 1, 1), and the right side computes the values of K for the vector beginning at grid point (1, 2, 1). The sequence for the steps of the calculation procedure, and the memory planes utilized in each step of the procedure, are listed in table 4. The total time to compute the right hand side of the Poisson equation is $34+24K+(1/2)(I_xJ_xK+I_xJ+J_xK+K_xI)$ clock periods. The operation count is $11 I_xJ_xK$, and the nominal operation rate is 440 MFLOPS.

The next step in the algorithm is to solve the Poisson equation for the pressure. As for the Helmholtz equation, the iterative procedures for solving the Poisson equation are discussed later in this section.

The last procedure for the first stage of the Runge-Kutta/Crank-Nicholson temporal discretization is to correct the velocity values using the updated values of the pressure. From the spatial discretization of Eq. (15), the three velocity correction equations may be written as

$$u_{ijk}^{n+1/2} = u_{ijk}^* - (p_{i+1jk}^{n+1/2} - p_{i-1jk}^{n+1/2})/2\Delta x \quad (25)$$

$$v_{ijk}^{n+1/2} = v_{ijk}^* - (p_{ij+1k}^{n+1/2} - p_{ij-1k}^{n+1/2})/2\Delta y \quad (26)$$

$$w_{ijk}^{n+1/2} = w_{ijk}^* - (dh_k p_{ijk+1}^{n+1/2} + dm_k p_{ijk}^{n+1/2} + dl_k p_{ijk-1}^{n+1/2}) \quad (27)$$

The 12 operation RALU pipeline for performing the velocity correction computations is illustrated in figure 14. In this figure, the memory planes indicated are for the computation of u , v , and w for the vectors beginning at grid point $(1, 1, 1)$. The sequence for the eight steps of the calculation procedure, and the memory planes utilized in each step, are listed in table 5. The startup time and actual computation time for the procedure are $120+48K$ and $IxJxK$ clock periods, respectively. Note that the step prior to the velocity correction procedure is the solution of the Poisson equation for pressure. Therefore, the boundary cache already contains the updated pressure values which are accessed as BPD data in the velocity correction procedure. As a result, there is no internode communication delay for this procedure.

Upon completion of the velocity correction procedure, the values for u , v , and w no longer reside in the memory planes in which they were stored initially. In order to simplify the implementation of subsequent procedures in the algorithm, these values are transferred back to their initial memory locations. As indicated in table 5, the values of u , which were initially stored in MPs 0-7, reside in MPs 8-15. The memory plane-to-memory plane transfers of these values, which are implemented by routing the vectors through MASNET, may be performed for all eight vectors simultaneously. It takes $IxJxK/8$ clock periods to perform this procedure. The values for v and w are treated in a similar manner. Combining this transfer time with the startup time and actual computation time, it takes $120+48K+1.375 IxJxK$ clock periods to implement the velocity correction procedure. The operation count for the procedure is $12 IxJxK$, and the nominal operation is 174 MFLOPS.

At this point in the algorithm, the primitive variables have been advanced to the $t=t^{n+1/2}$ time level. The next steps in the algorithm are to perform the computations for the second stage of the Runge-Kutta/Crank-Nicholson temporal discretization, to advance the solution to $t=t^{n+1}$. Comparison of the equations for the two stages reveals that the only differences occur in calculation of the advection term, as indicated by Eqs. (10) and (16). However, the additional term which appears in each of the three advection term component equations is easily treated with only minor modifications to the RALU pipelines. Hence, the calculation procedures for the second stage are nearly identical to those for the first stage, and will not be discussed here.

For advancement of the solution from $t=t^n$ to $t=t^{n+1}$, the time involved in computing the explicit terms is $522+348K+13.75 \text{ IxJxK}+6(\text{IxJ}+\text{JxK}+\text{KxI})$ clock periods. The operation count for these computations is 204 IxJxK . Neglecting the startup time and internode communication delay time, the sustained operation rate is around 297 MFLOPS.

Red-Black SOR Solution of the Helmholtz and Poisson Equations: Red-Black SOR is a two-color point method in which a point Jacobi type iterative scheme is utilized. A complete iteration entails updating red points ($i+j+k$ odd) first, and then updating black points ($i+j+k$ even) using the latest available values in calculating the residual. Considering the first Runge-Kutta stage of the algorithm, the Helmholtz equation for the x velocity component is

$$u^* - (h/4Re) \nabla^2 u^* = L^n \quad (28)$$

After second-order central differencing Eq. (28), the residual equation may be written as

$$\begin{aligned} (4Re/h)R_{ijk}^m &= (4Re/h)L_{ijk}^n + (u_{i+1jk}^v + u_{i-jk}^v)\Delta x^2 + (u_{ij+1k}^v + u_{ij-1k}^v)/\Delta y^2 \\ &+ Dh_k u_{ijk+1}^v + D\ell_k u_{ijk-1}^v - B_k u_{ijk}^m \end{aligned} \quad (29)$$

where

$$B_k = (4Re/h) + 2/\Delta x^2 + 2/\Delta y^2 - Dm_k ,$$

$v = m$ for red points, $v = m+1$ for black points, and m denotes the iteration level. Applying successive over-relaxation (SOR), the update equation becomes

$$u_{ijk}^{m+1} = u_{ijk}^m + \omega (4Re/h)R_{ijk}^m/B_k \quad (30)$$

where ω denotes the relaxation parameter.

The first step of the computational procedure is to update the values of u for the vector beginning at grid point (1, 1, 1), at red points only (i.e. grid points (1, 1, 1), (3, 1, 1), (5, 1, 1), etc.). Hence, consecutive values in the operand vectors are accessed from every other sequential memory location of the appropriate memory planes, rather than from consecutive memory locations. This gives vector lengths of $I \times J \times K/16$ for each step of the computational procedure. The 17 operation RALU pipeline for calculating the residuals, performing an in-line local convergence check, and updating the values of u is illustrated in figure 15. The memory planes indicated in the figure are for the first step of the procedure, where the values of u_{ijk}^m are accessed from MP0 and the values of u_{ijk}^{m+1} are stored in MP 15.

Subsequent steps in the computational procedure involve updating the values of u for the remaining red points, and then updating the values of u for the black points. The grid point at which the computations are begun for each step, and the memory planes utilized, are listed in table 6.

In order to simplify the implementation of subsequent iterations, the updated values of u are always transferred back to their initial memory locations. Hence, each step of the procedure also involves a memory plane-to-memory plane transfer of the updated values via MASNET, as indicated at the right of figure 15. The memory planes utilized in the intranode data transfers, and the sequence in which the transfers occur, are listed in the last two columns of table 6. As indicated, the values of u which are updated in the first step, are transferred back to their initial memory locations in the third step.

The iterative solution of this Helmholtz equation requires $2(IxJ+JxK+KxI)$ boundary-point values of u per interior Node. For the computational subdomains considered in this analysis, all of the BPD data may be stored within the boundary cache. As an iteration proceeds and values of u are updated, the updated boundary-point values are immediately routed to their destination Nodes, replacing the old values. For a global mapping of the computational grid as specified at the beginning of this section, the BPD data generated from updating u at red points is not used in the destination Nodes until black points are updated, and vice versa. For the sequence of computations specified in table 6, the time between updating a boundary-point value and utilizing that value in the destination Node is at least $IxJxK/4$ clock periods. Thus, the BPD data is pre-communicated long before it is required in

an ongoing computation of a destination Node, but is never communicated before the old values have been utilized. This not only ensures proper operation of the iterative method, but also ensures that there are no internode communication delays for the computational procedure.

The total computation time for performing a Red-Black SOR iteration, including the intranode transfer of all the updated values back to their initial memory locations, is $36+48K+(1+1/16)IxJxK$ clock periods. Assuming it takes h_r iterations to attain global convergence of the solution, the total computation time for solution of the Helmholtz equation is $h_r (36+48K+IxJxK) + 1/16(IxJxK)$ clock periods. The operation count for the procedure is $17 h_r IxJxK$. The nominal operation rate is 340 MFLOPS.

The procedures for solving the Helmholtz equations for the y and z velocity components are identical to that for the x velocity component, although different memory planes are utilized in the various steps. Solution of the Poisson equation is also quite similar, in that only minor modifications to the RALU pipeline must be made. After central-differencing Eq. (14), the residual and update equations for the Poisson equation are

$$R_{ijk}^m = (P_{i+1jk}^v + P_{i-1jk}^v)/\Delta x^2 + (P_{ij+1k}^v + P_{ij-1k}^v)/\Delta y^2 + D h_k P_{ijk+1}^v + D l_k P_{ijk-1}^v - B_k P_{ijk}^m - K_{ijk}^* \quad (31)$$

$$P_{ijk}^{m+1} = P_{ijk}^m + \omega R_{ijk}^m / B_k \quad (32)$$

where

$$B_k = 2/\Delta x^2 + 2/\Delta y^2 - D m_k$$

Comparison of Eqs. (31) and (32) with Eqs. (29) and (30) indicates that there is one less operation in the RALU pipeline for the Poisson equation. However, the computation time for performing an iteration is the same as that for the Helmholtz equation. Assuming that p_r iterations are required to attain global convergence of the solution, the total computation time for solving the Poisson equation is $p_r (36+48K+IxJxK)+(1/16)IxJxK$ clock periods, the operation count is $16 p_r \times IxJxK$, and the nominal operation rate is 320 MFLOPS.

ZEBRA 1 Solution of the Helmholtz and Poisson Equations: ZEBRA 1 is a two-color line method in which line SOR is performed along alternating vertical lines. The tridiagonal systems of equations generated for the vertical lines are first solved for red lines ($i+j$ even), and then for black lines ($i+j$ odd) using the latest available values in computing the residuals. Once again considering the Helmholtz equation for the x velocity component at the first Runge-Kutta stage, the residual equation for ZEBRA 1 may be written as

$$\begin{aligned} (4Re/h)R_{ijk}^m &= (4Re/h)L_{ijk}^n + (u_{i+1jk}^v + u_{i-1jk}^v)/\Delta x^2 + (u_{ij+1k}^v + u_{ij-1k}^v)\Delta y^2 \\ &+ Dh_k u_{ijk+1}^m - B_k u_{ijk}^m + D\ell_k u_{ijk-1}^m \end{aligned} \quad (33)$$

where

$$B_k = (4Re/h) + 2/\Delta x^2 + 2/\Delta y^2 - Dm_k ,$$

$v = m$ for red lines, and $v = m+1$ for black lines. After applying line SOR, the update equation, written in delta form, becomes

$$-D\ell_k \Delta u_{ijk-1}^m + (B_k/\omega) \Delta u_{ijk}^m - Dh_k \Delta u_{ijk+1}^m = (4Re/h) R_{ijk}^m \quad (34)$$

where

$$\Delta u_{ijk}^m = u_{ijk}^{m+1} - u_{ijk}^m$$

and ω denotes the relaxation parameter. Utilizing the Thomas algorithm to solve the tridiagonal systems of equations, a two-step procedure is developed for updating the velocity. In the first step

$$E_{ijk} = ((4Re/h) R_{ijk}^m + D\ell_k E_{ijk-1}) / (B_k/\omega - D\ell_k F_{ijk-1}) \quad (35)$$

$$F_{ijk} = Dh_k / (B_k/\omega - D\ell_k F_{ijk-1}) \quad (36)$$

are calculated from $k = 1$ up through $k = K$. In the next step

$$\Delta u_{ijk}^m = E_{ijk} + F_{ijk} \Delta u_{ijk+1}^m \quad (37)$$

$$u_{ijk}^{m+1} = u_{ijk}^m + \Delta u_{ijk}^m \quad (38)$$

are calculated from $k = K$ down through $k = 1$.

The computational procedure begins with the calculation of E and F for the vectors beginning at grid point $(1, 1, 1)$, at those grid points which lie along red lines. Consecutive values of the operand vectors are thus accessed from every other sequential memory location of the appropriate memory planes.

Subsequent steps in the procedure involve the calculation of E and F at grid points which lie along this same subset of red lines, moving up the grid in the vertical direction. Due to the recursive nature of the Thomas algorithm, the computations proceed up the grid in consecutive x-y planes. Hence, for the memory plane allocation procedure utilized herein, the computations may only be vectorized accross x-y planes, and the vector lengths are $I \times J/8$. Upon completion of the calculation of E and F at all grid points which lie along this subset of red lines, Δu^m and u^{m+1} are calculated along the same lines, proceeding down the grid in the vertical direction. Again, the vector lengths for this procedure are $I \times J/8$. Upon completion of these computations, u will have been updated along one-fourth of the red lines.

The 22 operation RALU pipeline for calculating the residual, performing an in-line local convergence check, and computing the values of E and F is illustrated in figure 16. The memory planes indicated are for the vectors beginning at grid point (1, 1, k). The sequence of steps in the procedure for computing E and F for this subset of red lines, and the memory planes utilized in each step, are listed in table 7. As indicated at the right of the figure and in the last two columns of table 7, the computational procedure also involves an intranode data transfer of the values of u_{ijk}^m to a memory plane for temporary storage. This memory plane-to-memory plane transfer is performed so that in the second step of the procedure, where values of u_{ijk}^{m+1} are calculated, the updated values may be stored in their respective initial memory locations.

The next step in the procedure entails computing Δu^m and u^{m+1} at grid points along the subset of red lines for which E and F are calculated in the

previous step. The 9 operation RALU pipeline for performing this calculation is illustrated in figure 17. In this pipeline, the computations are performed in 3 consecutive x-y planes, simultaneously. Beginning at the left of the figure, values of Δu_{ijk+3}^m are routed to the RALU for use in the computation of Δu_{ijk+2}^m . The computed values of Δu_{ijk+2}^m then branched off to two independent pipelines. In the left pipeline, the values of u_{ijk+2}^{m+1} are computed and then routed to memory; in the right pipeline the computations for Δu_{ijk+1}^m are begun. This process is repeated twice in order to compute u_{ijk+1}^{m+1} and u_{ijk}^{m+1} . As indicated at the bottom right of the figure, the values of Δu_{ijk}^m are routed out through MASNET and temporarily stored in the vector delay and regeneration unit. In the subsequent step, these values are then routed back to the RALU for use in calculating Δu_{ijk-1}^m . The sequence of procedures for updating u for this subset of red lines, and the memory planes utilized in each step, are listed in table 8. Note that the sequence of calculations are repeated every 12 x-y planes.

At this point in the computations, the values of u will have been updated at the grid points for one-fourth of the red lines. Subsequent steps in the procedure entail updating the values of u for the remaining red lines, and then for black lines.

For the previously specified mapping of the computational grid into the Nodes, where $K = N/4$, the grid points which lie along a given vertical line are mapped into four different Nodes. For the ZEBRA 1 computational procedure described above where the computations proceed up, and then down the grid in consecutive x-y planes, three of the four Nodes will be idle at any given time. In order to avoid this situation, a different mapping of the computa-

tional grid is utilized in which the subdomains of dimension $I \times J \times K$ have dimensional lengths of $I = L/16$, $J = M/8$, and $K = N$. To further simplify the implementation of the procedures, K is assumed to be a multiple of 12. Conceptually, the computational domain is now mapped into a two-dimensional lattice of Nodes, where all grid points along a given vertical line are mapped into the same Node. The hypercube network then provides direct links between all adjacent Nodes in the lattice, and for interior Nodes, the number of boundary-point values generated for each primitive variable is $2(I \times K + J \times K)$, rather than $2(I \times J + J \times K + K \times I)$. This new mapping of the computational domain has little effect on the computational procedures for the explicit terms, particularly since the Nodal memory plane allocation procedure remains unchanged.

As with the Red-Black SOR iterative method, the internode communication requirements for the ZEBRA 1 method are such that there are no internode communication delays for this computational procedure. Assuming it takes h_z iterations to attain global convergence of the solution, the total computation time for solution of the Helmholtz equation is $h_z(1600 + 4I \times J)K/3$ clock periods. The operation count is $25 h_z I \times J \times K$, and the nominal operation rate is 375 MFLOPS.

The procedures for solving the Helmholtz equations for the y and z velocity components are identical to that for the x velocity component. Solution of the Poisson equation is also quite similar, although there is one less operation in the calculation of the residual equation. Assuming that p_z iterations are required to attain global convergence of the solution, the total computation time for solving the Poisson equation is $p_z(1600 + 4I \times J)K/3$, the operation count is $24 p_z I \times J \times K$, and the nominal operation rate is 360 MFLOPS.

Projected Timing Results: The actual computation time for advancing the solution from time t^n to t^{n+1} is dependent upon the size of the grid subdomains, the relative dimensions of the subdomains, and the iterative method used to solve the Helmholtz and Poisson equations. The size of the grid subdomains which may be mapped into the Nodes depends upon the effective number of variables which must be stored per grid point, including the temporary storage of intermediate results. For the computational procedures detailed herein, both the Red-Black SOR and ZEBRA 1 algorithms require the effective storage of 12 variables per grid point.

The relative dimensions of the $L \times M \times N$ computational grid are dictated by the directional resolution requirements for the particular problem to be solved. For a true spatially developing flow, the projected resolution requirements suggest dimensions of length $L \approx 4 \times N$ and $M \approx 2 \times N$. Considering a computational grid of dimension $3360 \times 1680 \times 960$, which contains roughly 5.4×10^9 grid points, then the grid subdomains for the Red-Black SOR and ZEBRA 1 algorithms are of dimension $420 \times 420 \times 240$ and $210 \times 210 \times 960$, respectively. Storing 12 variables per grid point, subdomains of this size require about 508 Mwords of storage, utilizing about 99.2% of the available Nodal storage capacity.

The projected timing results for simulation of the true spatially developing flow are presented in table 9. For the explicit terms, comparison of the actual operation rate with the nominal operation rate indicates that the internode communication delay and startup costs constitute less than 0.5% of the computation time. For solution of the Helmholtz and Poisson equations, the startup costs for the Red-Black SOR and ZEBRA 1 methods constitute less

than 0.1% and 1% of the computation time, respectively. Timing results for the Helmholtz and Poisson equation solutions are given as a function of the number of iterations required to attain global convergence of the solution to some specified accuracy. The accuracy to which the solutions must be resolved, and the number of iterations required to attain this accuracy, must still be resolved through numerical experimentation. However, for a computational grid of this size the spectral radius for both Red-Black SOR and ZEBRA 1 approaches 1, so both iterative methods require on the order of thousands of iterations to attain global convergence of the solutions. Hence, the sustained operation rate for the algorithm approaches the sustained operation rate of the Helmholtz and Poisson equation solvers, which is roughly 76% and 83% of the Nodal peak speed for the Red-Black SOR and ZEBRA 1 algorithms, respectively. The sustained operation rate for a full 128 Node NSC is projected to be about 43 GFLOPS and 47 GFLOPS, respectively.

Despite the large operation rates at which the computations are performed, this simulation requires a substantial amount of computation time. Assuming that the ZEBRA 1 algorithm only requires 1000 iterations to attain global convergence of the solution for each Helmholtz and Poisson equation, which is quite an optimistic assumption, the time required for advancing the solution one time step is nearly 6 1/2 hours. Since a full simulation requires thousands of time steps, the real time for performing these computations in a dedicated environment is on the order of months. Although the resolution demands for many laminar-turbulent transition problems require far fewer grid points than the 5.4 billion utilized in this example, this result indicates that in order to perform this simulation in a reasonable

amount of time, iterative techniques with vastly greater convergence rates than those for Red-Black SOR and ZEBRA 1 must be employed.

One means for enhancing the convergence rates for the Helmholtz and Poisson equation solvers is to utilize multigrid methods [12, 13] in conjunction with suitable single grid iterative schemes. Incorporation of the multigrid methods into the iterative schemes involves the implementation of additional computational procedures, which involve the projection of the values onto coarser grids and then interpolation of the values back to finer grids. The projection and interpolation procedures appear to be well suited for implementation on the NSC.

In order to approximate the performance of a multigrid method for performing this simulation, the following assumptions are made for incorporating a multigrid method into the ZEBRA 1 algorithm. The effective spectral radius for a full multigrid V-cycle is projected to be around 0.5. Defining convergence of the solution to be achieved when the residual has been reduced by 10^{-5} , 18 V-cycles are required to attain convergence. It is further assumed that the computational work for performing one multigrid V-cycle is equivalent to that for performing two ZEBRA 1 iterations on the finest grid, and that there are no additional internode communication delays. Then for the simulation on a 3360x1680x960 grid, the total time to advance the solution one time step is projected to be around 14 minutes.

A second means for enhancing the convergence rates of the Helmholtz and Poisson equation solvers is to utilize conjugate gradient methods [14, 15] in conjunction with the single grid iterative methods. Incorporation of the multigrid and conjugate gradient methods into the algorithm, and implementation of those algorithms on the NSC, are the subject of future work.

Concluding Remarks

The NSC is a multi-purpose parallel-processing supercomputer which is being developed to provide an efficient means for simulating large, numerically intensive, scientific problems. Rapid solution of these problems is attained by structuring the computational procedures of the solution algorithms to effectively utilize both the fine grain and coarse grain parallelism inherent in the NSC architecture. The objective of this paper has been to present a detailed description of the procedures involved in implementing one such algorithm on the NSC.

Crucial to the effective utilization of the fine grain parallelism is the choice of a memory plane allocation procedure for storing the array elements of the various variables. The allocation procedure utilized in this analysis is fairly complicated in that a lot of "bookkeeping" is required in order to keep track of where, and in what order, the primitive variables and intermediate results are stored. Less complicated allocation procedures could have been used, such as storing all the array elements of a particular variable within a single memory plane. For most of the computational procedures outlined in this analysis, use of this simpler allocation procedure would not affect significant changes in the RALU pipeline configurations, and only a slight degradation in the operation rates would be realized. However, the changes which would be required for implementing the ZEBRA 1 iterative solution of the Helmholtz and Poisson equations would result in about a 30% decrease in the operation rate. Conversely, allocation procedures for which

more efficient RALU pipelines can be configured, could have been utilized. However, the modest improvement in the operation rates were deemed insufficient to justify the significant increase in complexity for implementing the procedures.

Utilization of the coarse grain parallelism involves the distribution of the computations over multiple Nodes. For the finite difference algorithm considered here, in which fairly simple iterative methods and a Cartesian grid with grid stretching in only one coordinate direction are utilized, the computational grid is mapped into either a two or three-dimensional lattice of Nodes, and the Nodes perform nearly identical processes. Furthermore, for a given computational procedure, less than 2% of the data need be communicated between the Nodes. By explicitly defining this boundary-point data and precommunicating the data before it is required in the destination Nodes, most of the internode communication overhead for implementing this algorithm has been eliminated.

The projected timing results for implementing this algorithm on the NSC indicate that operation rates at around 75% of the machine peak speed are attainable. For a 128 Node NSC, the projected operation rates would be in excess of 42 GFLOPS. However, the timing results also indicate that for computational grids of the size which can be accommodated on the NSC, the convergence rates for the Red-Black SOR and ZEBRA 1 algorithms are inadequate for performing this simulation in a reasonable amount of time. In fact, the convergence rates for any of the single grid iterative methods are likely to be inadequate. Consequently, a more desirable approach appears to be the use of conjugate gradient or multigrid methods in conjunction with the iterative techniques.

Acknowledgements

The authors would like to thank Daniel Nosenchuck and Michael Littman at Princeton University for providing detailed information about the NSC architecture, and for providing invaluable input during many enlightening discussions. Valuable support was also provided by Douglas Dwoyer at NASA Langley Research Center.

Research was supported under NASA Cooperative Agreement NCC1-24 while the first author was in residence at the Joint Institute for Advancement of Flight Sciences, NASA Langley Research Center.

References

1. Nosenchuck, D. M. and Littman, M. G.: "The Coming of Age of the Parallel-Processing Supercomputer," presented at the 23rd Annual Space Conference, April 22, 1986.
2. Nosenchuck, D. M. and Littman, M. G.: "The Navier-Stokes Computer," Presented at the Symposium on Future Directions of Computational Mechanics, ASME, December 7, 1986.
3. Zang, T. A. and Hussaini, M. Y.: "Numerical Experiments on the Stability of Controlled Shear Flows," AIAA Paper No. 85-1698.
4. Orzag, S. A. and Kells, L. C.: "Transition to Turbulence in Plane Poiseuille and Plane Couette Flows," Journal of Fluid Mechanics, Vol. 96, January 1980, pp. 159-205.
5. Kleiser, L. and Schumann, U.: "Spectral Simulation of the Laminar-Turbulent Transition Process in Plane Poiseuille Flow," Proc. of ICASE Symposium on Spectral Methods, (R. Voight, D. Gottlieb, and M. Y. Hussaini, eds.), SIAM-CBMS, 1984, pp. 141-163.
6. Brooks, E. et al: "Nearest Neighbor Concurrent Processor," DOE Research and Development Report, 1981, CALT-68-687.

7. Fasel, H.: "Investigation of Stability of Boundary Layers by a Finite-Difference Model of the Navier-Stokes Equations", Journal of Fluid Mechanics, Vol. 78, May 1976, pp. 355-383.
8. Patera, A. T.: "A Spectral Element Method for Fluid Dynamics: Laminar Flow in a Channel Expansion," Journal of Computer Physics, Vol. 54, 1984, pp. 468-488.
9. Zang, T. A. and Hussaini, M. Y.: "On Spectral Multigrid Methods for the Time-Dependent Navier-Stokes Equations," Appl. Math. Comput., vol. 19, 1986, p. 359-372.
10. Street, C. L. and Hussaini, M. Y.: "Finite Length Effects in Taylor-Couette Flow," ICASE Report No. 86-59, NASA CR-178175, September 1986.
11. South, J. C.; Keller, J. D.; and Hafez, M. M.: "Vector Processor Algorithms for Transonic Flow Calculations," AIAA Journal, July 1980, Article No. 79-1457R, pp. 787-792.
12. Holter, W. H.: "A Vectorized Multigrid Solver for the Three-Dimensional Poisson Equation." Proc. 2nd Copper Mountain Conference on Multigrid Methods, Copper Mountain, Co., March 31 - April 3, 1985.
13. Cline, K. Schaffer, S., and Slaughter, G.: "A Multigrid Approach for the Three Dimensional Heat Equation with Discontinuous Coefficients," Harvey Mudd College Internal Report, 1984.
14. Reid, J. K.: "On the Method of Conjugate Gradients for the Solution of Large Sparse Systems of Linear Equations," Large Sparse Sets of Linear Equations, J. K. Reid (ed.), Academic Press, New York, pp. 231-254, 1971.
15. Concus, P., Golub, G. H., and Oleary, D. P.: A Generalized Conjugate Gradient Method for the Numerical Solution of Elliptic Partial Differential Equations, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose (eds.), Academic Press, New York, pp. 309-332, 1976.

STEP # In Procedure	1st GRID POINT OF VECTOR	Memory Planes In Which Values Are Stored					
		OPERANDS			RESULTS		
		u Values	v Values	w Values	η_{ijk}	ζ_{ijk}	F_{ijk}
1	(1,1,1)	1,3,4,0,6	8	2	9	11	12
3	(1,2,1)	2,0,5,1,7	9	3	10	12	13
5	(1,3,1)	3,1,6,2,4	10	0	11	13	14
7	(1,4,1)	0,2,7,3,5	11	1	12	14	15
9	(1,1,2)	5,7,2,4,0	12	6	13	15	8
11	(1,2,2)	6,4,3,5,1	13	7	14	8	9
13	(1,3,2)	7,5,0,6,2	14	4	15	9	10
15	(1,4,2)	4,6,1,7,3	15	5	8	10	11

TABLE 1: Sequence of calculations for the first step in the advection term calculation procedure.

STEP # In Procedure	1st GRID POINT OF VECTOR	Memory Planes In Which Values Are Stored						
		OPERANDS					RESULTS	
		u_{ijk}	v Values	w Values	η_{ijk}	ζ_{ijk}	G_{ijk}	H_{ijk}
2	(1,1,1)	0	12,8,14	3,2,1	9	11	4	13
4	(1,2,1)	1	13,9,15	0,3,2	10	12	5	14
6	(1,3,1)	2	14,10,12	1,0,3	11	13	6	15
8	(1,4,1)	3	15,11,13	2,1,0	12	14	7	8
10	(1,1,2)	4	10,12,8	7,6,5	13	15	0	9
12	(1,2,2)	5	11,13,9	4,7,6	14	8	1	10
14	(1,3,2)	6	12,14,10	5,4,7	15	9	2	11
16	(1,4,2)	7	13,15,11	6,5,4	8	10	3	12

TABLE 2: Sequence of calculations for the second step in the advection term calculation procedure.

STEP # In The Procedure	1st GRID POINT OF VECTOR	Memory Planes In Which Values Are Stored			
		OPERANDS			RESULTS
		u_{ijk}	u @ 6 neighboring points	F_{ijk}	L_{ijk}
1	(1,1,1)	0	0,1,3,4,6	12	8
2	(1,2,1)	1	1,2,0,5,7	13	9
3	(1,3,1)	2	2,3,1,6,4	14	10
4	(1,4,1)	3	3,0,2,7,5	15	11
5	(1,1,2)	4	4,5,7,2,0	8	12
6	(1,2,2)	5	5,6,4,3,1	9	13
7	(1,3,2)	6	6,7,5,0,2	10	14
8	(1,4,2)	7	7,4,6,1,3	11	15

TABLE 3: Sequence of calculations for the right-hand side of the Helmholtz equation for the x velocity component.

STEP # In The Procedure	1st GRID POINT OF VECTOR	Memory Planes					1st GRID Point of VECTOR	Memory Planes				
		OPERANDS			RESULTS K_{ijk}	OPERANDS			RESULTS K_{ijk}			
		u Values	v Values	w Values		u Values		v Values		w Values		
1	(1,1,1)	0	9,11	6,2,4	15	(1,2,1)	1	10,8	7,3,5	14		
2	(1,3,1)	2	11,9	4,0,6	13	(1,4,1)	3	8,10	5,1,7	12		
3	(1,1,2)	4	13,15	0,6,2	11	(1,2,2)	5	14,12	1,7,3	10		
4	(1,3,2)	6	15,13	2,4,0	9	(1,4,2)	7	12,14	3,5,1	8		

TABLE 4: Sequence of calculations for the right-hand side of the Poisson equation.

1st GRID POINT OF VECTOR	Memory Planes In Which Values Are Stored								
	OPERANDS		RESULTS	OPERANDS		RESULTS	OPERANDS		RESULTS
	P Values	u_{ijk}^*	$u_{ijk}^{n+1/2}$	P Values	v_{ijk}^*	$v_{ijk}^{n+1/2}$	P Values	w_{ijk}^*	$w_{ijk}^{n+1/2}$
(1,1,1)	10	0	15	11,9	8	7	14,10,12	2	13
(1,2,1)	11	1	14	8,10	9	6	15,11,13	3	12
(1,3,1)	8	2	13	9,11	10	5	12,8,14	0	15
(1,4,1)	9	3	12	10,8	11	4	13,9,15	1	14
(1,1,2)	14	4	11	15,13	12	3	8,14,10	6	9
(1,2,2)	15	5	10	12,14	13	2	9,15,11	7	8
(1,3,2)	12	6	9	13,15	14	1	10,12,8	4	11
(1,4,2)	13	7	8	14,12	15	0	11,13,9	5	10

TABLE 5: Sequence of calculations for the velocity correction procedure.

1st GRID POINT OF VECTOR	Memory Planes In Which Values Are Stored					
	OPERANDS			RESULTS	MP to MP transfer	
	u_{ijk}^m	u @ 6 neighboring points	L_{ijk}	u_{ijk}^{m+1}	From	To
(1,1,1)	0	0,1,3,4,6	8	15	9	5
(2,2,1)	1	1,2,0,5,7	9	14	8	4
(1,3,1)	2	2,3,1,6,4	10	13	15	0
(2,4,1)	3	3,0,2,5,7	11	12	14	1
(1,4,2)	7	7,4,6,1,3	15	11	13	2
(2,3,2)	6	6,7,5,0,2	14	10	12	3
(1,2,2)	5	5,6,4,3,1	13	9	11	7
(2,1,2)	4	4,5,7,2,0	12	8	10	6
(2,1,1)	0	0,1,3,4,6	8	15	9	5
(1,2,1)	1	1,2,0,5,7	9	14	8	4
(2,3,1)	2	2,3,1,6,4	10	13	15	0
(1,4,1)	3	3,0,2,5,7	11	12	14	1
(2,4,2)	7	7,4,6,1,3	15	11	13	2
(1,3,2)	6	6,7,5,0,2	14	10	12	3
(2,2,2)	5	5,6,4,3,1	13	9	11	7
(1,1,2)	4	4,5,7,2,0	12	8	10	6

TABLE 6: Sequence of calculations for the Red-Black SOR solution of the Helmholtz equation for the x velocity component.

1st GRID POINT OF VECTOR	Memory Planes In Which Values Are Stored								
	OPERANDS					RESULTS		MP to MP transfer	
	u_{ijk}^m	u @ neighboring points	L_{ijk}	E_{k-1}	F_{k-1}	E_k	F_k	From	To
(1,1,k)	0	0,1,3,4,6	8	15	12	9	10	0	5
(1,1,k+1)	4	4,5,7,2,0	12	9	10	13	14	4	3
(1,1,k+2)	2	2,3,1,6,4	10	13	14	11	8	2	7
(1,1,k+3)	6	6,7,5,0,2	14	11	8	15	12	6	1
(1,1,k+4)	0	0,1,3,4,6	8	15	12	9	10	0	5
(1,1,k+5)	4	4,5,7,2,0	12	9	10	13	14	4	3
(1,1,k+6)	2	2,3,1,6,4	10	13	14	11	8	2	7
(1,1,k+7)	6	6,7,5,0,2	14	11	8	15	12	6	1
(1,1,k+8)	0	0,1,3,4,6	8	15	12	9	10	0	5
(1,1,k+9)	4	4,5,7,2,0	12	9	10	13	14	4	3
(1,1,k+10)	2	2,3,1,6,4	10	13	14	11	8	2	7
(1,1,k+11)	6	6,7,5,0,2	14	11	8	15	12	6	1

TABLE 7: Sequence of calculations for E and F in the ZEBRA 1 solution of the Helmholtz equation for the x velocity component.

x-y Plane	Memory Planes In Which Values Are Stored					
	OPERANDS			RESULTS		
	u, E, & F @ k+2	u, E, & F @ k+1	u, E, & F @ k	u_{ijk+2}^{m+1}	u_{ijk+1}^{m+1}	u_{ijk}^{m+1}
k+9	1, 15, 12	7, 11, 8	3, 13, 14	6	2	4
k+6	5, 9, 10	1, 15, 12	7, 11, 8	0	6	2
k+3	3, 13, 14	5, 9, 10	1, 15, 12	4	0	6
k	7, 11, 8	3, 13, 14	5, 9, 10	2	4	0

TABLE 8: Sequence of calculations for updating u in the ZEBRA 1 solution of the Helmholtz equation for the x velocity component.

Projected Timing Results for a True Spatially Developing Flow			
Computational Procedure	Parameter	Iterative Method	
		Red-Black SOR	ZEBRA 1
	Subdomain Dimensions	420x420x420	210x210x960
Explicit Terms	Computation Time (sec.)	29.17	29.24
	Operation Rate (MFLOPS)	296.5	295.3
	Nominal Operation Rate	296.7	296.7
Helmholtz and Poisson Equation Solvers	Computation Time (sec.)	$1.06 + 4.23(p_r + 3h_r)$	$5.70(p_z + 3h_z)$
	Operation Rate	334.9	367.9
	Nominal Operation Rate	335.0	371.2
	Total Computation Time	$30.23 + 4.23(p_r + 3h_r)$	$29.24 + 5.70(p_z + 3h_z)$

TABLE 9: Projected Timing Results for a True Spatially Developing Flow on a 3360x1680x960 Computational Grid.

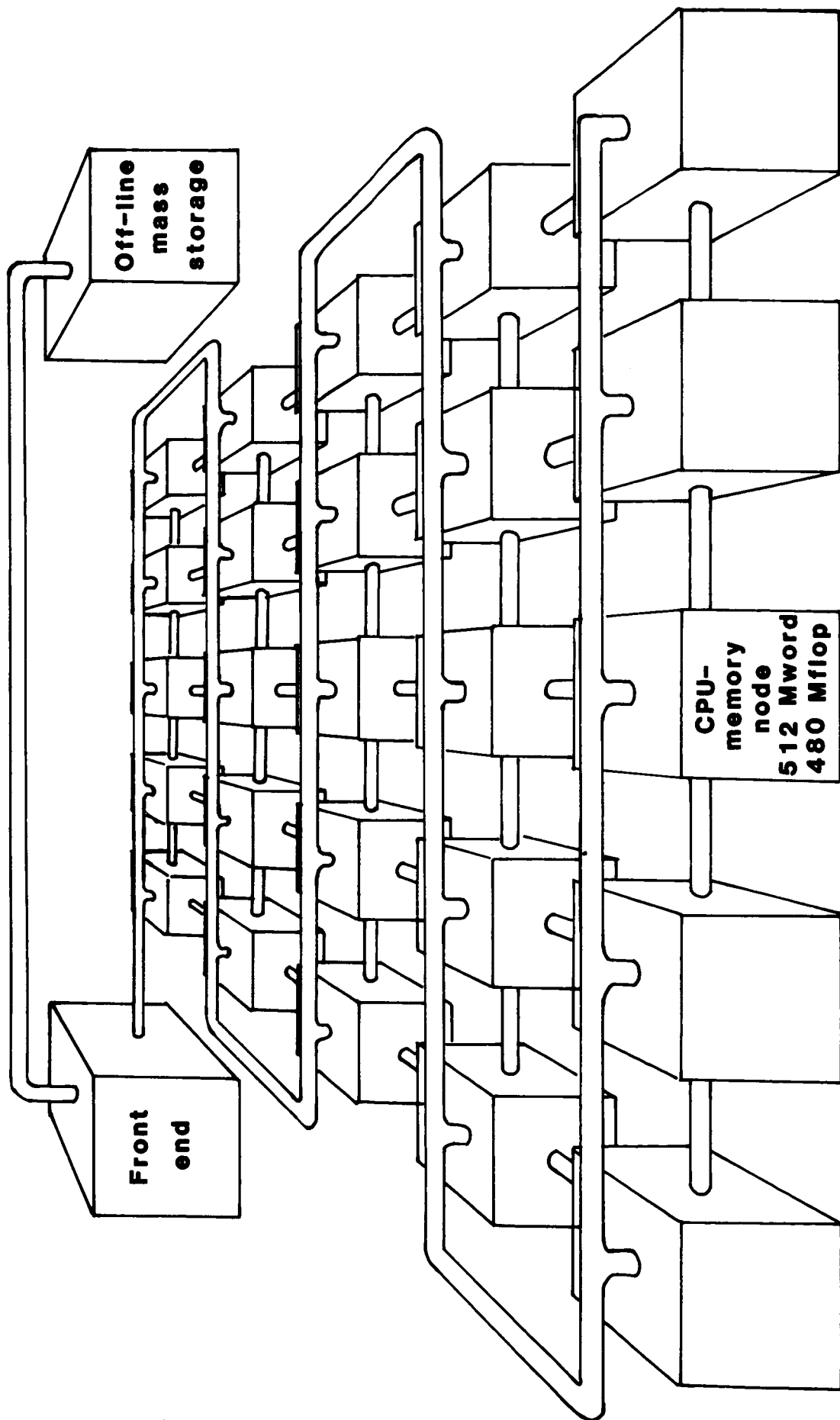


Figure 1.- Overall layout of the Navier-Stokes computer.

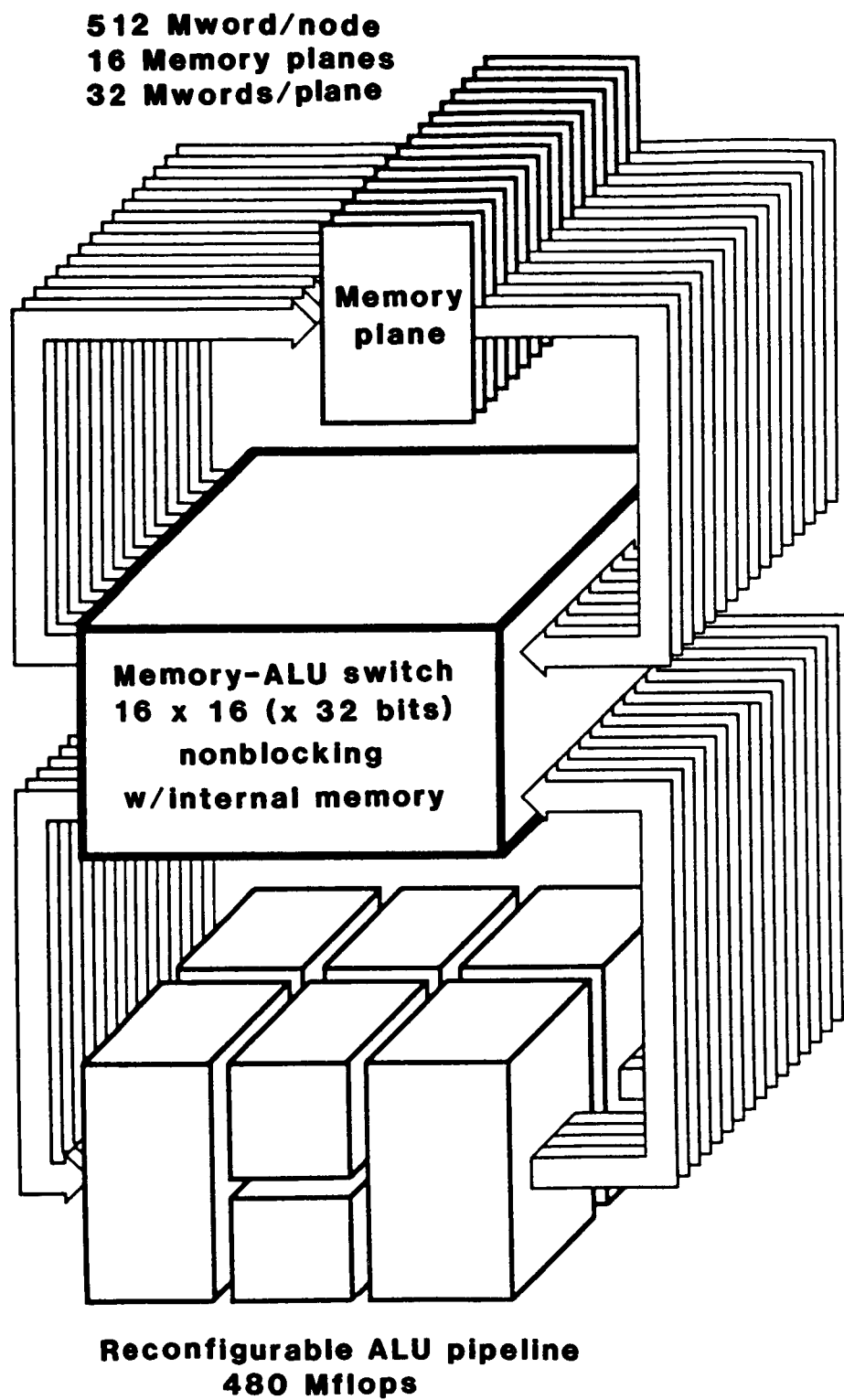


Figure 2.- Layout of memory /MASNET/RALU interconnects.

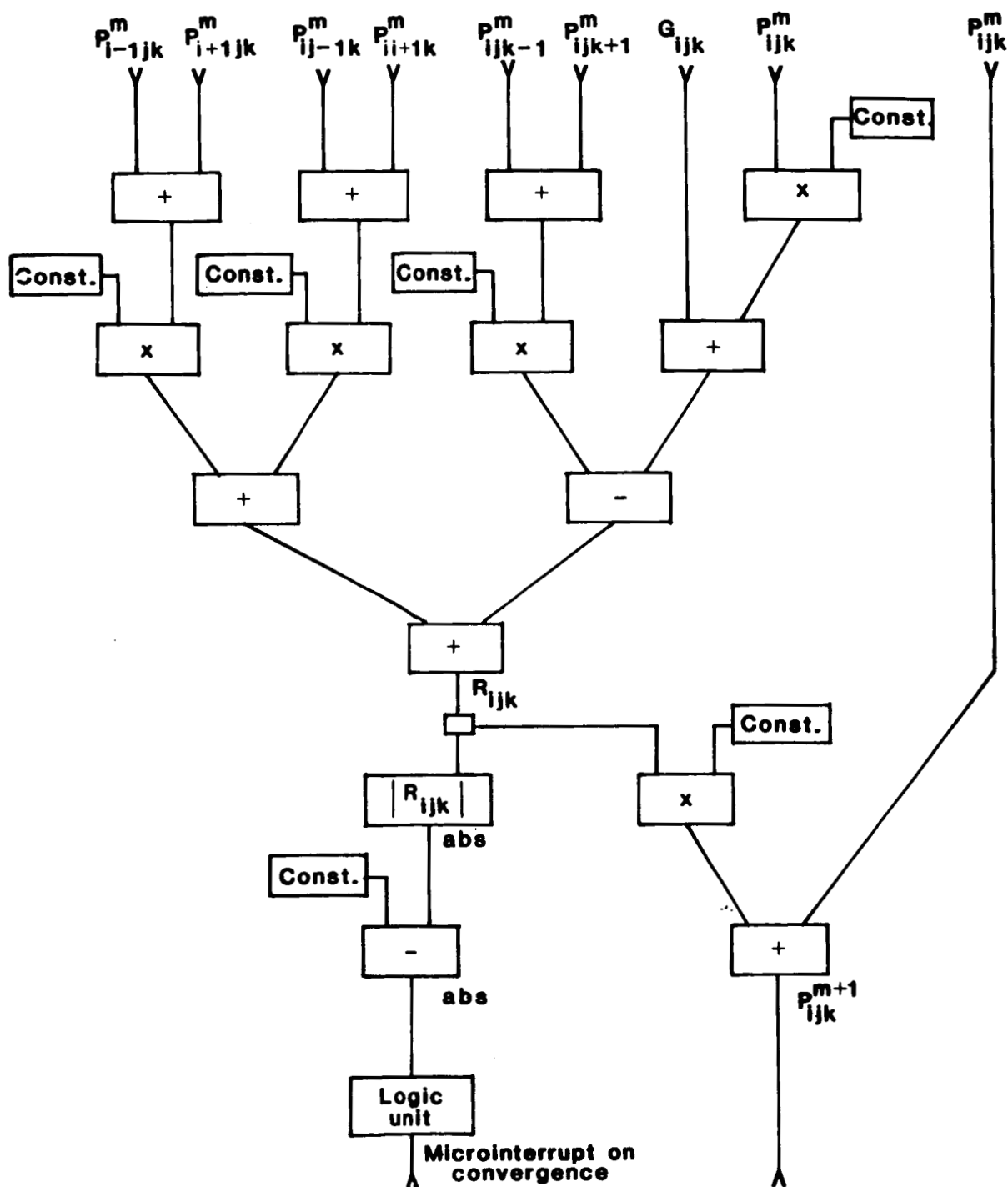


Figure 3.- Block diagram of pipeline for performing a point Jacobi iteration of the 3-D Poisson equation. The operation performed by each floating-point processing element is indicated by +, -, or x. Constant parameter latches are only indicated for elements in which the constant is utilized for this procedure.

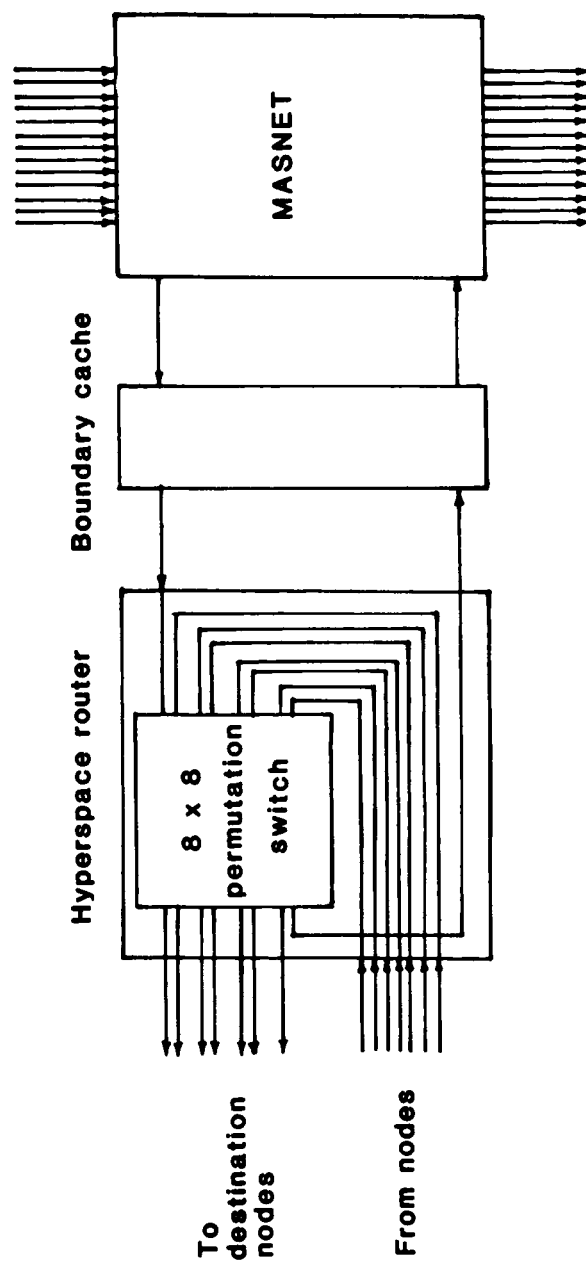


Figure 4.- Block diagram of MASNET/boundary cache/hyperspace router interconnects.

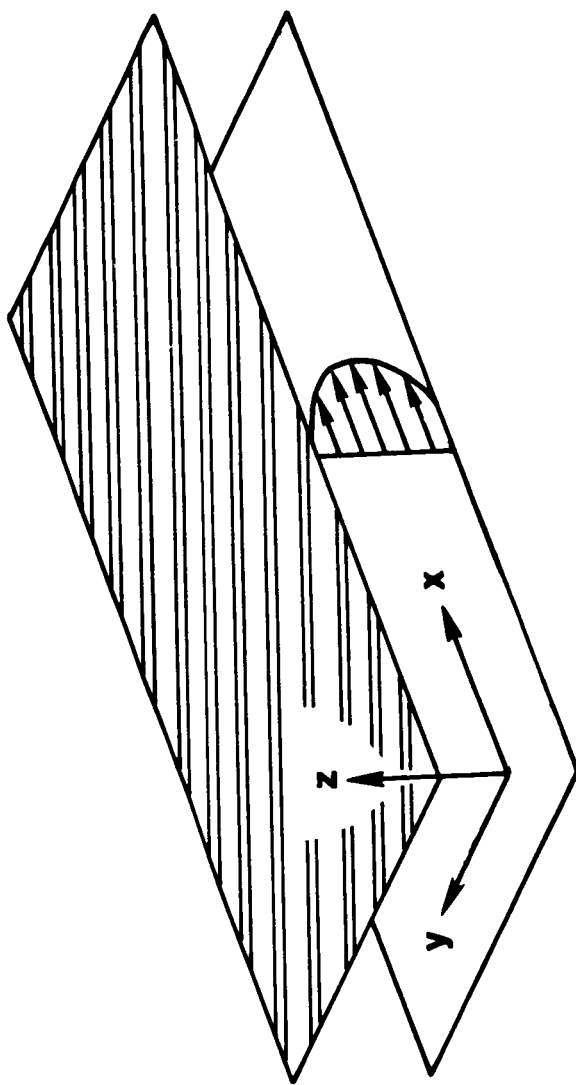


Figure 5.- Channel flow geometry.

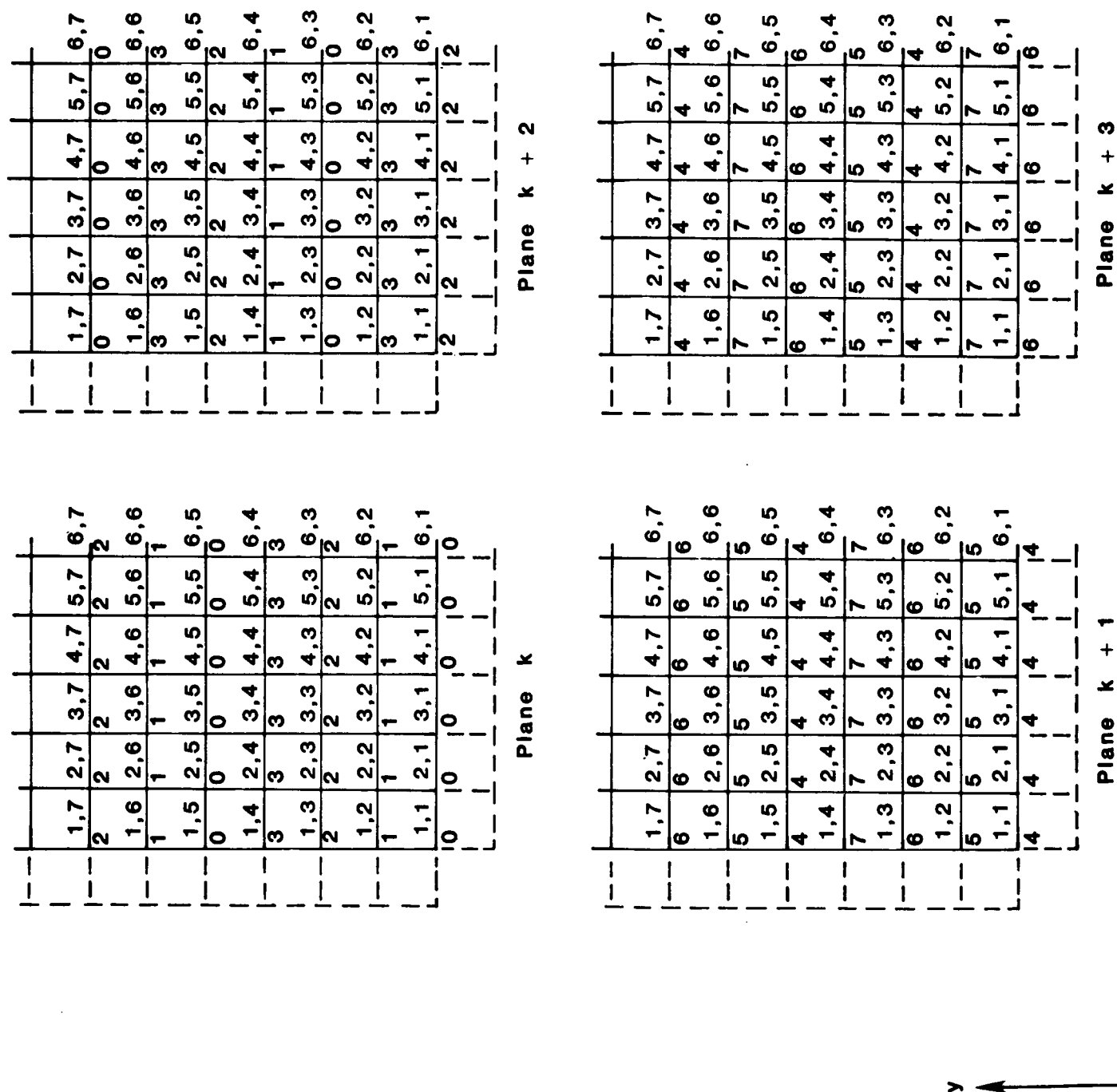


Figure 6.- Schematic of the memory plane allocation procedure for the primitive variable u .

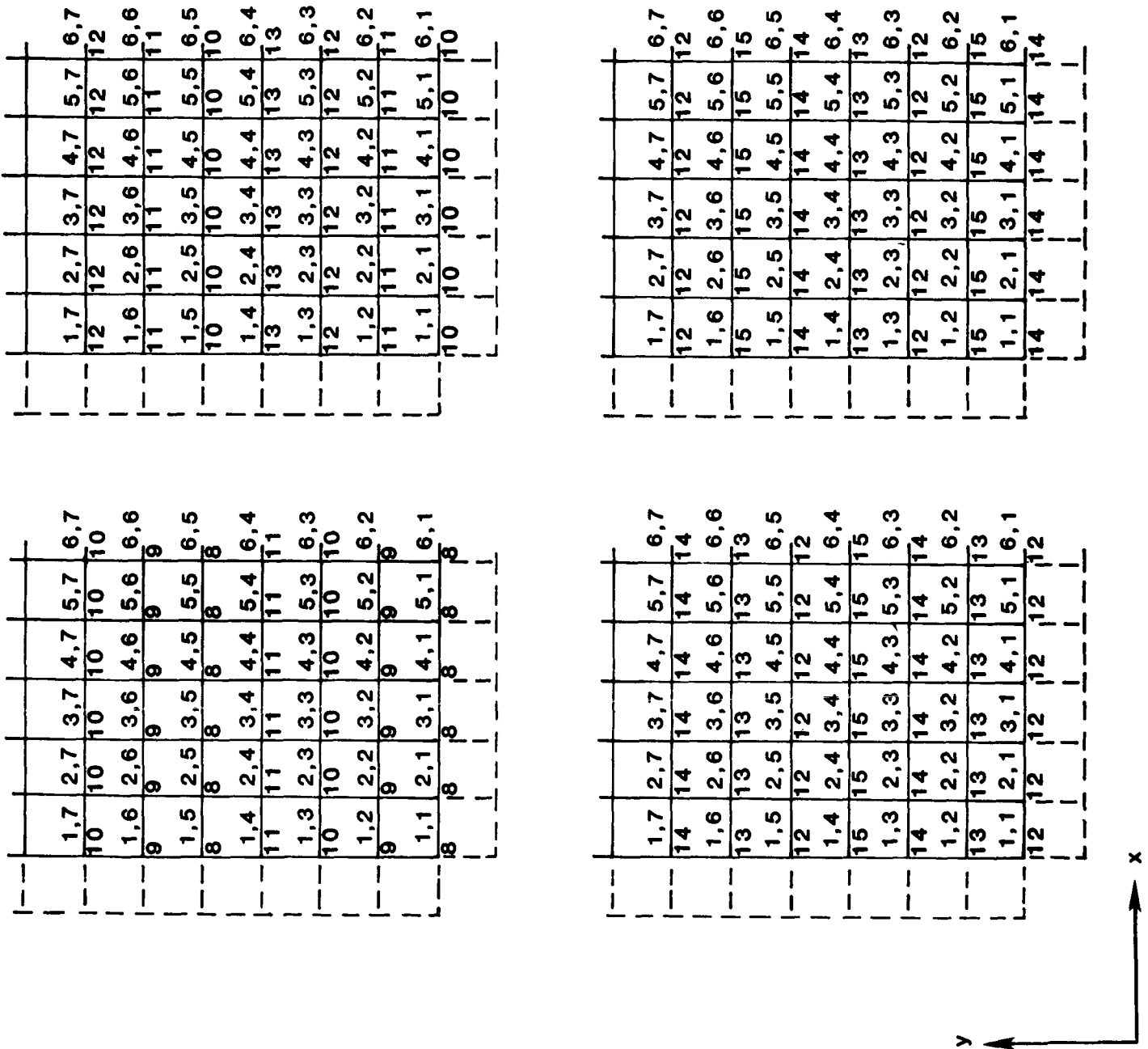


Figure 7.- Schematic of the memory plane allocation procedure for the primitive variable v .

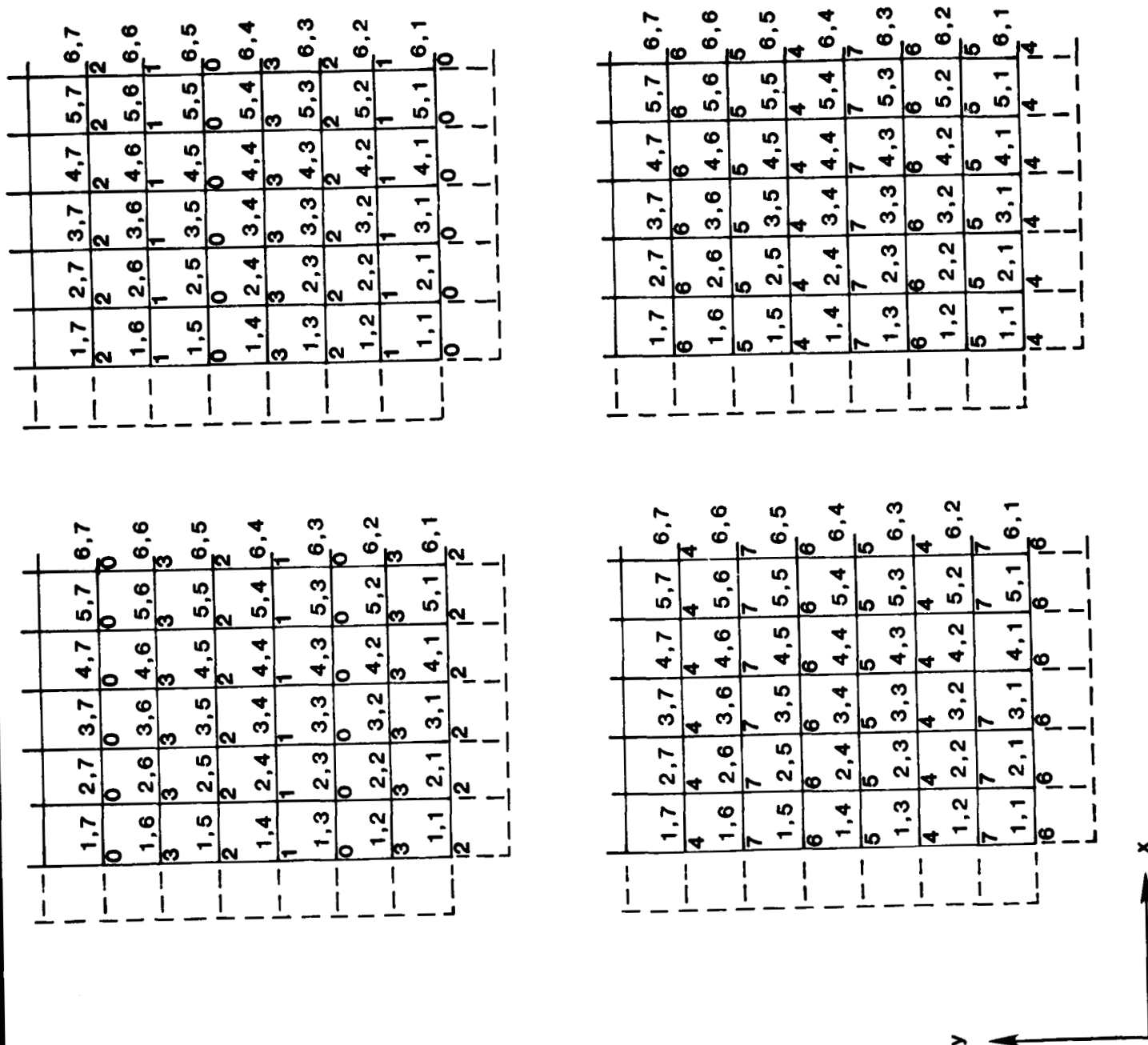
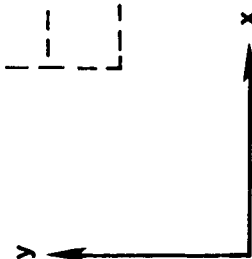


Figure 8.-- Schematic of the memory plane allocation procedure for the primitive variable w .

	1,7	2,7	3,7	4,7	5,7	6,7			
	10	10	10	10	10	10			
	1,6	2,6	3,6	4,6	5,6	6,6			
		9	9	9	9	9			
	1,5	2,5	3,5	4,5	5,5	6,5			
	8	8	8	8	8	8			
	1,4	2,4	3,4	4,4	5,4	6,4			
	11	11	11	11	11	11			
	1,3	2,3	3,3	4,3	5,3	6,3			
	10	10	10	10	10	10			
	1,2	2,2	3,2	4,2	5,2	6,2			
	9	9	9	9	9	9			
	1,1	2,1	3,1	4,1	5,1	6,1			
	8	8	8	8	8	8			

	1,7	2,7	3,7	4,7	5,7	6,7			
	14	14	14	14	14	14			
	1,6	2,6	3,6	4,6	5,6	6,6			
	13	13	13	13	13	13			
	1,5	2,5	3,5	4,5	5,5	6,5			
	12	12	12	12	12	12			
	1,4	2,4	3,4	4,4	5,4	6,4			
	15	15	15	15	15	15			
	1,3	2,3	3,3	4,3	5,3	6,3			
	14	14	14	14	14	14			
	1,2	2,2	3,2	4,2	5,2	6,2			
	13	13	13	13	13	13			
	1,1	2,1	3,1	4,1	5,1	6,1			
	12	12	12	12	12	12			

Figure 9.- Schematic of the memory plane allocation procedure for the primitive variable P.



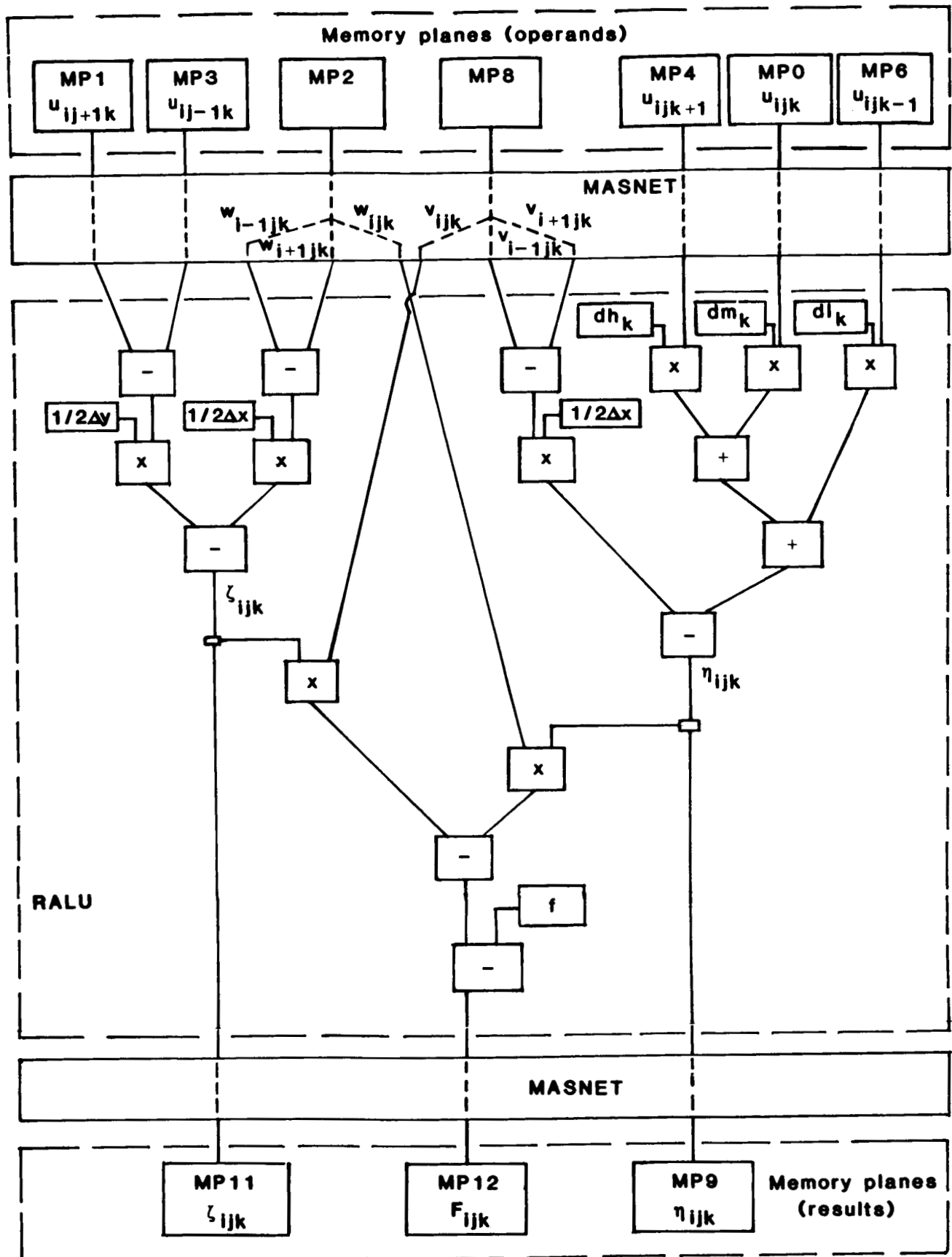


Figure 10.- Block diagram of pipeline for the first step in the advection term calculation procedure.

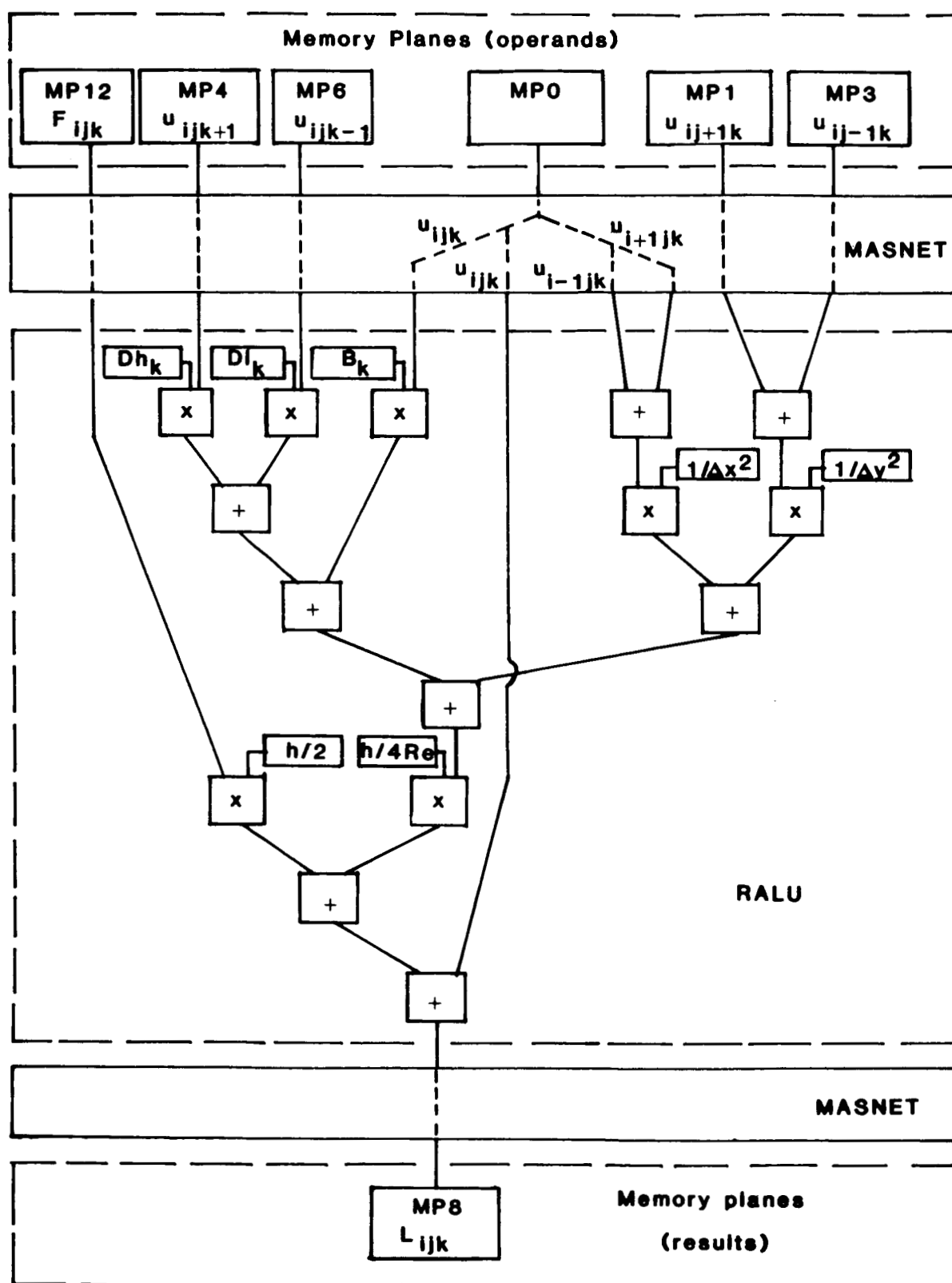


Figure 12.- Block diagram of pipeline for calculation of the right-hand side of the Helmholtz equation for the x velocity component.

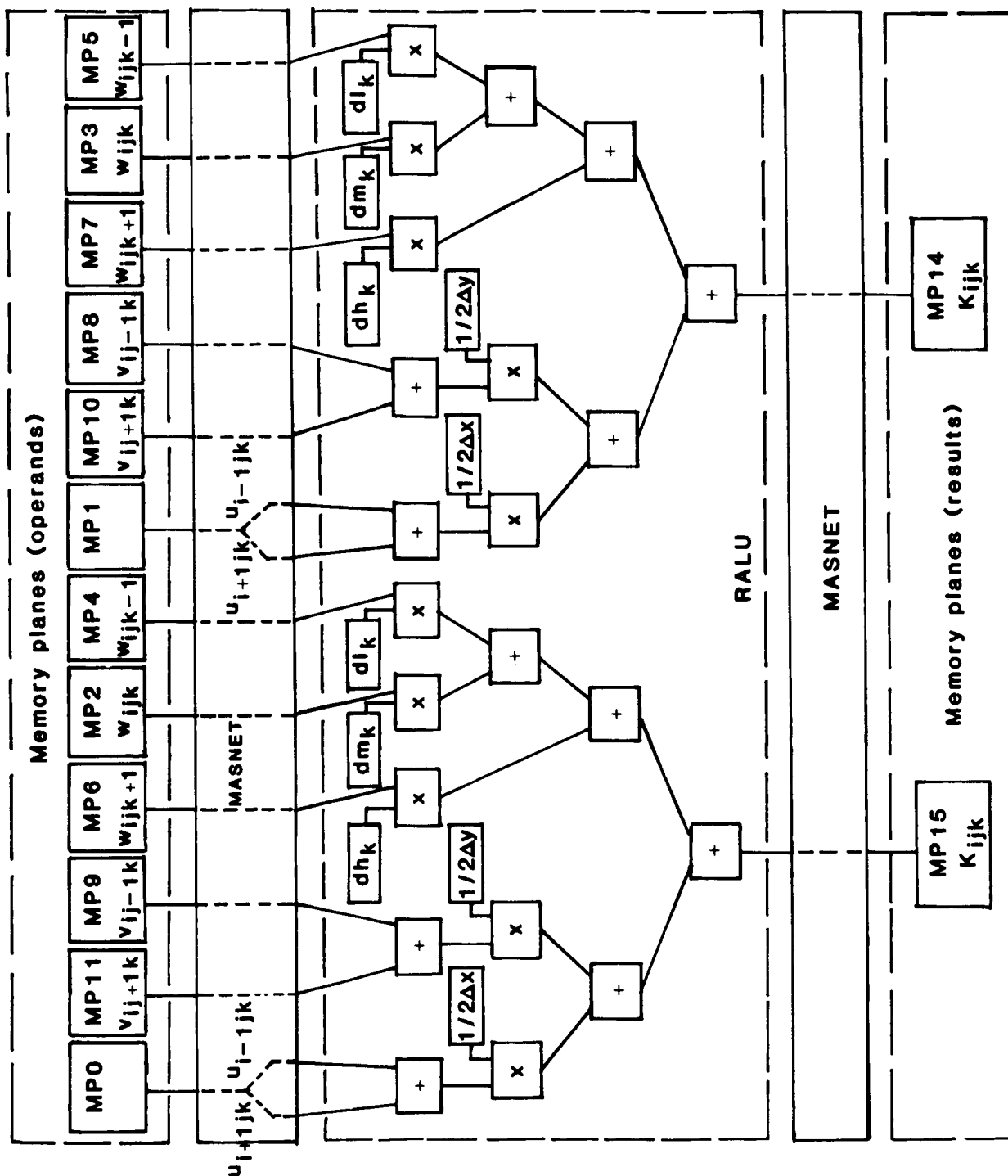


Figure 13.- Block diagram of pipeline for calculation of the right-hand side of the Poisson equation.

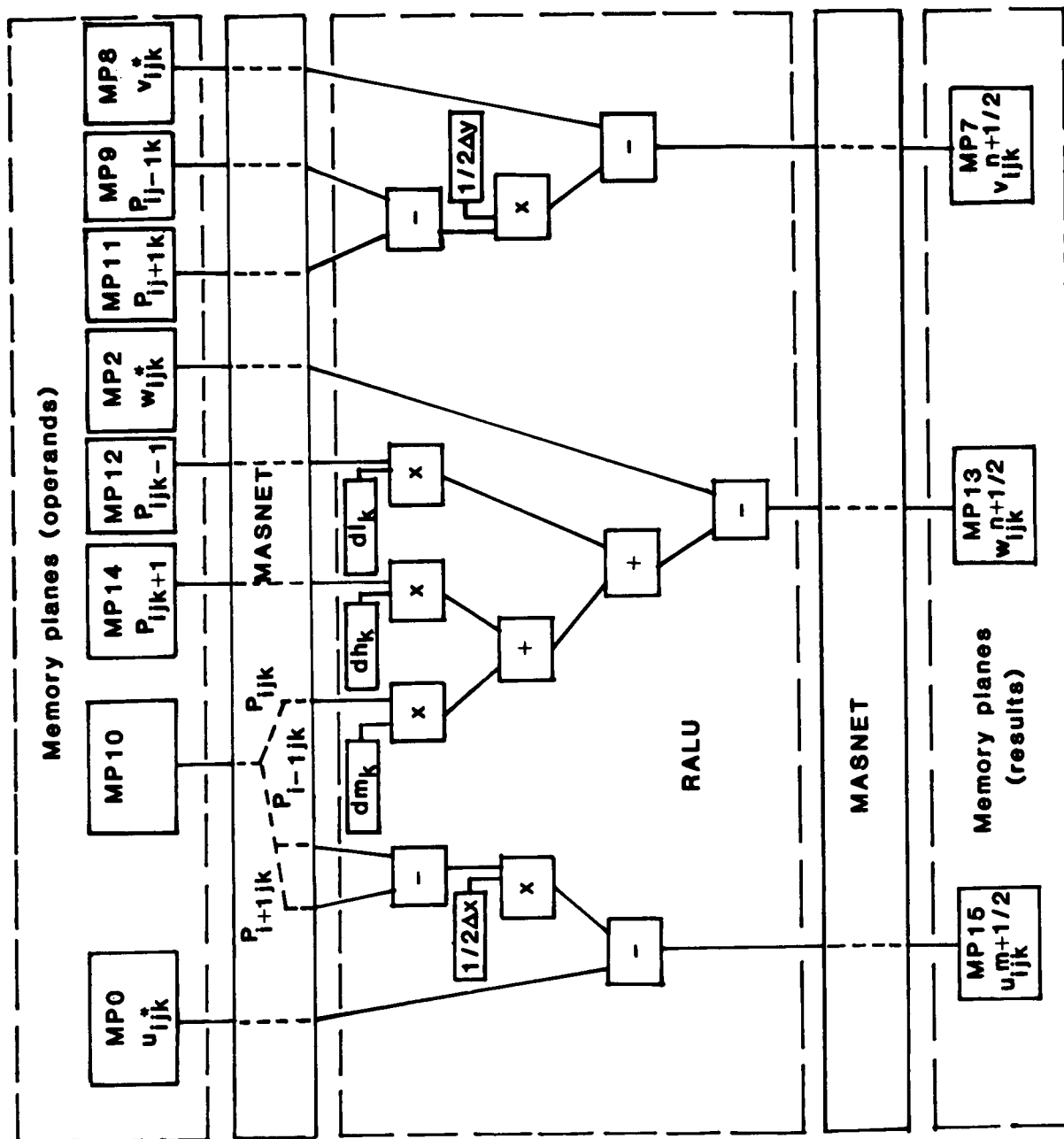


Figure 14.- Block diagram of pipeline for the velocity correction procedure.

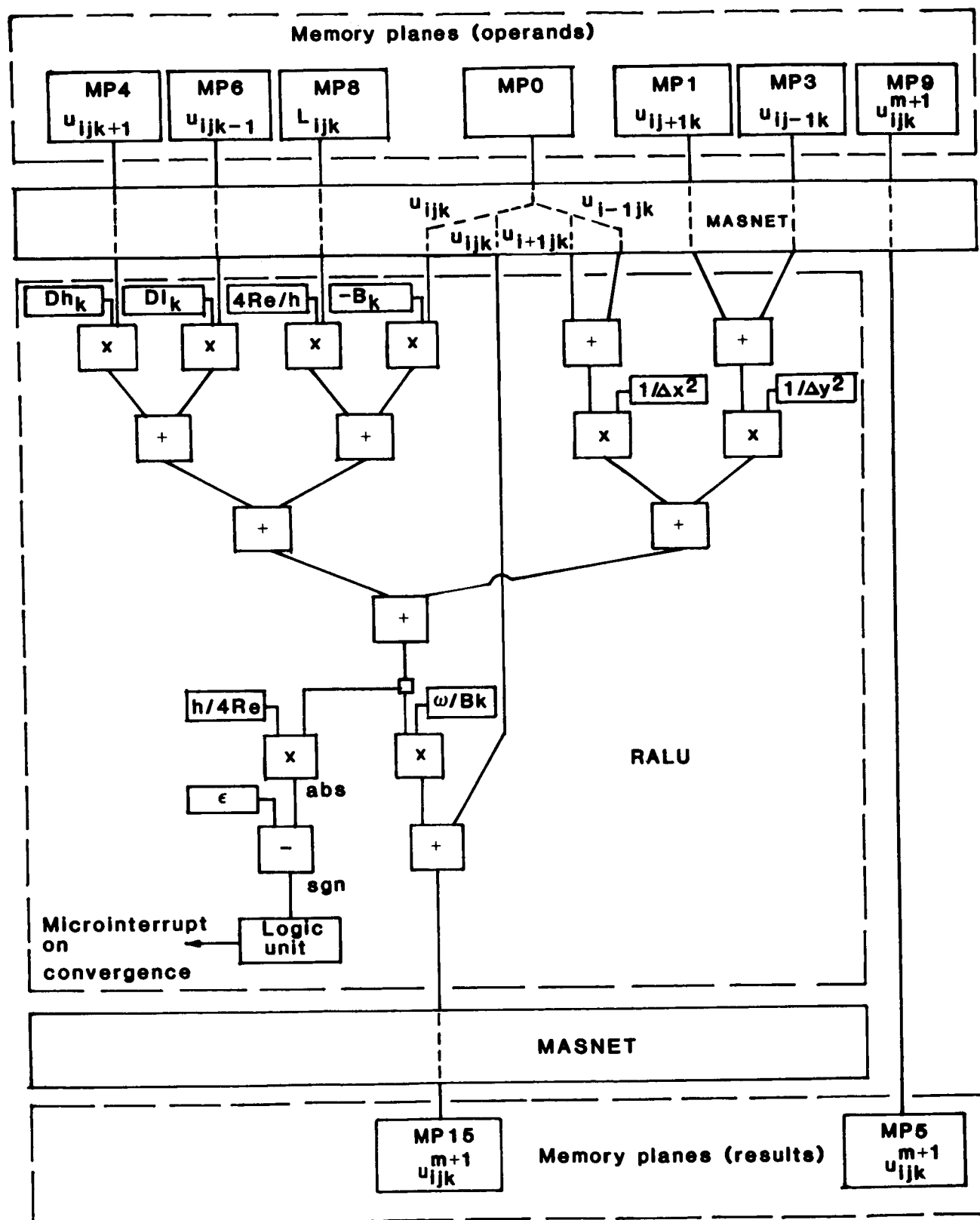


Figure 15.- Block diagram of pipeline for the red-black SOR iteration procedure.

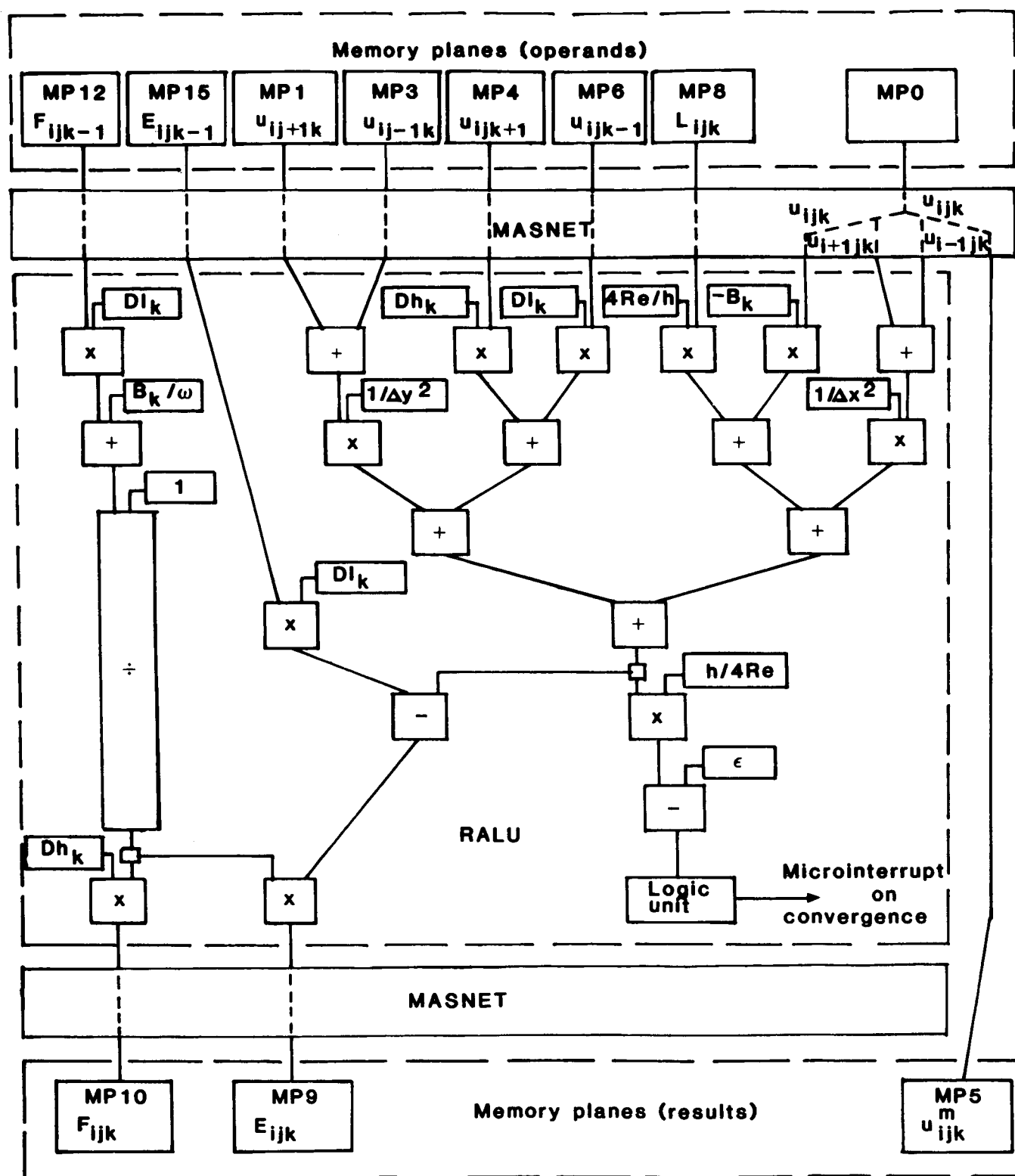


Figure 16.- Block diagram of pipeline for the first step in the ZEBRAL iteration procedure.

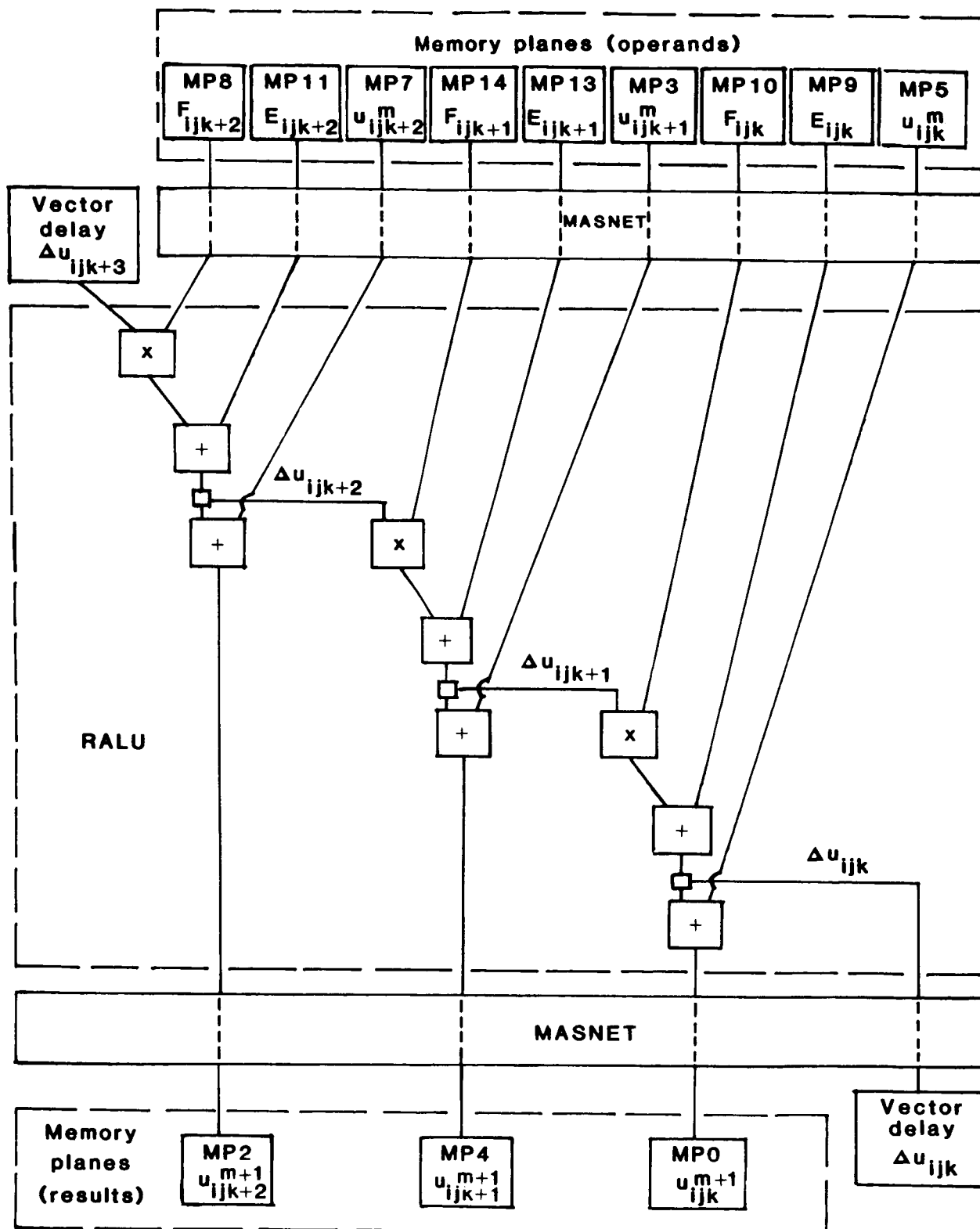


Figure 17.- Block diagram of pipeline for the second step in the ZEBRA1 iteration procedure.

Standard Bibliographic Page

1. Report No. NASA TM-89119		2. Government Accession No.		3. Recipient's Catalog No.	
4. Title and Subtitle Algorithm Implementation on the Navier-Stokes Computer				5. Report Date March 1987	
				6. Performing Organization Code	
7. Author(s) Steven E. Krist Thomas A. Zang				8. Performing Organization Report No.	
				10. Work Unit No. 505-60-01-02	
9. Performing Organization Name and Address NASA Langley Research Center Hampton, VA 23665-5225				11. Contract or Grant No.	
				13. Type of Report and Period Covered Technical Memorandum	
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546				14. Sponsoring Agency Code	
15. Supplementary Notes Steven E. Krist, The George Washington University, Joint Institute for Advancement of Flight Sciences, Hampton, Virginia. Thomas A. Zang, Langley Research Center, Hampton, Virginia.					
16. Abstract The Navier-Stokes Computer is a multi-purpose parallel-processing supercomputer which is currently under development at Princeton University. It consists of multiple local memory parallel processors, called Nodes, which are interconnected in a hypercube network. Details of the procedures involved in implementing an algorithm on the Navier-Stokes computer are presented in this paper. The particular finite-difference algorithm considered in this analysis was developed for simulation of laminar-turbulent transition in wall-bounded shear flows. Projected timing results for implementing this algorithm indicate that operation rates in excess of 42 GFLOPS are feasible on a 128 Node machine.					
17. Key Words (Suggested by Authors(s)) Parallel-processing Laminar-turbulent transition				18. Distribution Statement Unclassified - Unlimited Subject Category 02	
19. Security Classif.(of this report) Unclassified		20. Security Classif.(of this page) Unclassified		21. No. of Pages 74	
				22. Price A04	

For sale by the National Technical Information Service, Springfield, Virginia 22161